NASA/TM–1999-208792

SAGE
The Self-Adaptive Grid codE
Version 3

Carol B. Davies
Raytheon ITSS, Moffett Field, California

Ethiraj Venkatapathy
Thermoscience Institute, Eloret, Sunnyvale, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

August 1999

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

TABLE OF CONTENTS

THE SELF-ADAPTIVE GRID CODE, SAGE
Version 3

Carol B. Davies* and Ethiraj Venkatapathy†
Ames Research Center

SUMMARY

The previous version of the SAGE code (Version 2) is described in NASA TM-110350 (Davies and Venkatapathy, 1995). This new report on Version 3 includes all the information in the original publication plus all upgrades and changes to the SAGE code since that time. The most significant upgrade is the feature to move the outer grid boundary, either reducing or expanding the grid, based on flow features or other information. This ability was originally featured in a separate code called OUTBOUND developed by the same authors and extensively used in-house. In addition, a new reclustering algorithm has been added that permits very tight clustering in the boundary layer, a significant asset for grids used in viscous flow calculations.

The first section of this document describes the formulation of the adaption method. The second section is presented in the form of a user guide that explains the input and execution of the code, while the third section provides many examples. A special case is the ability to handle the PLOT3D BLANK option that is used in complex multigrid structures to blank out regions of overlapping grids. Due to its complexity and size, this option is not available in SAGEv3 but is described in this document in Section 5 and the special SAGE version can be obtained from the authors.

Successful application of the SAGE code in both two and three dimensions for the solution of various flow problems has proven the code to be robust, portable, and simple to use. Although the basic formulation follows the method of Nakahashi and Deiwert (1985), many modifications have been made to facilitate the use of the self-adaptive grid method for complex grid structures. Modifications to the method and the simple but extensive input options make this a flexible and user-friendly code. The SAGE code can accommodate two-dimensional and three-dimensional, finite-difference and finite-volume, single grid and zonal-matching multiple grid flow problems.

1. METHOD

1.1 Introduction

Solution-adaptive grid methods have become useful tools for efficient and accurate flow predictions. In supersonic and hypersonic flows, strong gradient regions such as shocks, contact discontinuities, and shear layers require careful distribution of grid points to minimize grid error and thus produce accurate flow-field predictions. Frequently, the generation of the first grid topology does not adequately capture these flow structures. It has been shown that an effective way of obtaining accurate solutions is by intelligently redistributing (i.e., adapting) the original grid points based on the initial flow-field solution and then computing a new solution using the adapted grid. The cost efficiency of grid adaptions in terms of CPU time depends on the basic formulation of the adaptive-grid solver. A short historical review and a list of references on the adaptive grid procedure used in the SAGE code are given in Davies and Venkatapathy (1991).

The self-adaptive grid procedure outlined by Nakahashi and Deiwert (1985) has evolved into a flexible tool for adapting and restructuring both two-dimensional (2-D) and three-dimensional (3-D) grids. The adaptive-grid method is based on grid-point redistribution through local error minimization. The procedure is analogous to applying tension and torsion spring forces proportional to the local flow gradient at every grid point and finding the equilibrium position of the resulting system of grid points. The multidimensional problem of grid adaption is split into a series of one-dimensional (1-D) problems along the computational coordinate lines. The reduced 1-D problem then requires a tridiagonal solver to find the location of grid points along a given coordinate line. Multidirectional adaption is achieved by the sequential

* Raytheon ITSS, Moffett Field, CA

† Thermoscience Institute, Eloret, Sunnyvale, CA

application of the method in each coordinate direction. This approach produces an extremely CPU-efficient algorithm.

The tension forces direct the redistribution of points to the strong gradient regions. The torsion forces relate information between the families of lines adjacent to one another, to maintain smoothness and a measure of orthogonality of grid lines. These smoothness and orthogonality constraints are direction-dependent, since they relate only the coordinate lines that are being adapted to the neighboring lines that have already been adapted. This implies that the solutions are non-unique and depend on the order and direction of adaption.

The Self-Adaptive Grid codE (SAGE) code has been built with many flexible elements that make it user-friendly. The second section of this report is a user guide, which is independent of the first section and can be used as a separate document. However, the nomenclature and details of the grid adaption procedure within the code consistently follow the analysis, allowing the user to understand the code and implement any individual changes, if desired. The user guide describes the input parameters and their effects as well as the adaption procedure, execution commands, and subroutines, and provides many examples.

1.2 Formulation of Adaptive-Grid Scheme

As stated in the introduction, adaption takes place as a series of one-dimensional adaptions. Figure 1 illustrates this concept: three constant k planes of an initial grid are shown in Fig. 1(a) and a flow-field solution has been obtained using this unadapted grid. The points in this grid are then adapted to the flow solution, starting on the first line $(j=1)$ on the lower plane $k=1$. In Fig. 1(b), the first plane has already been adapted and the second plane is the current adaption plane. The current adaption line (j) is shown, with previous lines already adapted and subsequent lines awaiting adaption. The third plane is still in its original form. Adaption is performed in this line-by-line, plane-by-plane manner until all requested planes are complete. It is then possible to perform an adaption in a second direction, adapting on top of the already adapted grid. The number and order of adaptions are arbitrary and depend on the type of flow problem and the purpose of the adaption.

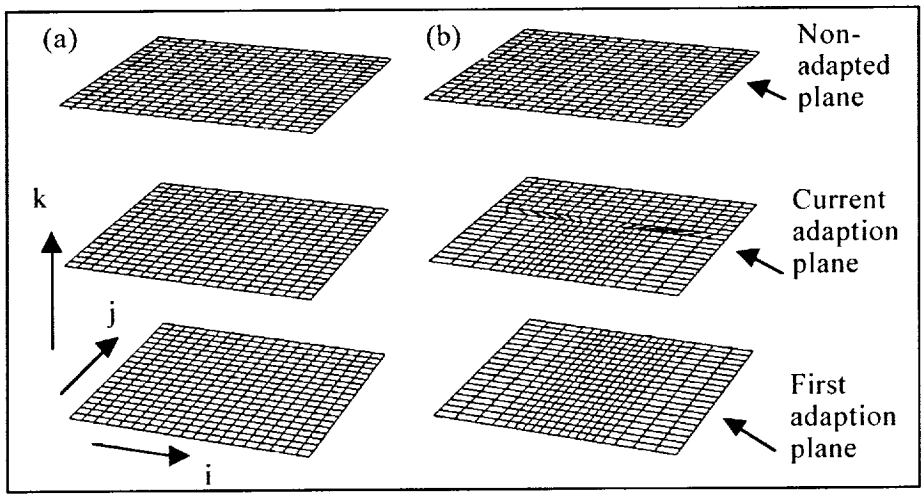

Figure 1. 3-D adaption. (a) Initial grid; (b) directional adaption.

Figure 2 shows a segment of the current adaption line in more detail. The lower plane has already been adapted and the upper plane is currently being adapted. Four forces control the redistribution of points along each line: the two tension springs that act on each side of a node, and the two torsion springs that control the smoothness of the grid. The tension forces ω cluster the redistributed points into the high-gradient regions. The torsion forces (τ and ψ) restrain the tension forces to maintain continuity between sequentially adapted lines. As shown in Fig. 2, the τ force acts from the previously adapted line within the current plane and the ψ force acts from the previously adapted line in the preceding plane.

A 3-D grid is described in terms of its computational coordinate directions (i,j,k) and its physical coordinates (x,y,z). An adaption can take place in any or all of the computational coordinate directions and the SAGE code permits any combination. However, for clarity, this analysis assumes that all adaptions are performed in the i direction and that stepping (within a plane surface) occurs in the j direction. Thus k is the marching direction for plane stepping. The current adaption line is thus j in the plane k where lines $j-1$, $j-2$, etc., have already been adapted. In addition, this report uses the term 'plane' to mean any 3-D coordinate surface.

The first step in the formulation of the adaption algorithm is to consider the adaption of a single line, with no torsional constraints (i.e., no smoothness control). Figure 3 shows a 2-D surface, where the arc length at A (i.e., $s_{i,j,k}$, with $s_{1,j,k} = .0$) along the current adaption line, j, is defined as

$$s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2} \qquad (1)$$

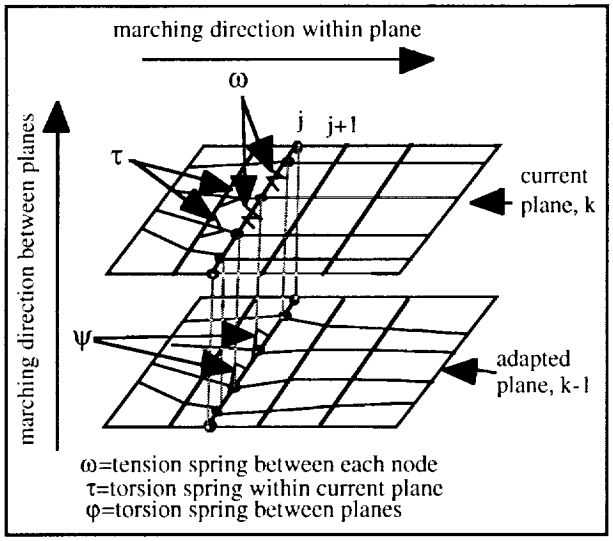

Figure 2. Line-by-line adaption showing tension and torsion forces.

A tension force, $f(\omega_i)$, is defined to act between each i and $i+1$ node such that

$$\omega_i \Delta s_i = K \qquad (2)$$

where ω, the tension constant, is a weighting function based on the flow gradient; $\Delta s_i = s_{i+1} - s_i$; and K is the resultant force. To redistribute the points along a line with the minimum solution error, the adaptive procedure defines the weighting function as proportional to the derivative of the flow variable (Nakahashi and Deiwert 1985). In this formulation, ω is defined as a function of the normalized flow gradient, $\bar{f}$, such that

$$\omega_i = 1 + A\bar{f}_i^B \qquad (3)$$

where A and B are constants directly related to the grid spacing and are chosen to maintain the grid intervals to within the limits (Δs_{MIN} and Δs_{MAX}) set by the user on input. A and B are defined in Appendix I of this section.

Equation (2) is written for each node on the line, giving a 1-D formulation that can be solved directly for Δs_i. Taking the sum of both sides of Eq. (2) produces

$$\sum_{l=1}^{n_i} \Delta s_l = s_{max} = K \sum_{l=1}^{n_i} \frac{1}{\omega_l}$$

giving

$$K = s_{max} \Big/ \sum_{l=1}^{n_i} \frac{1}{\omega_l} \qquad (4)$$

Substituting back in Eq. (2), we obtain

$$\Delta s_i = s_{max} \Big/ \omega_i \sum_{l=1}^{n_i} \frac{1}{\omega_l} \qquad (5)$$

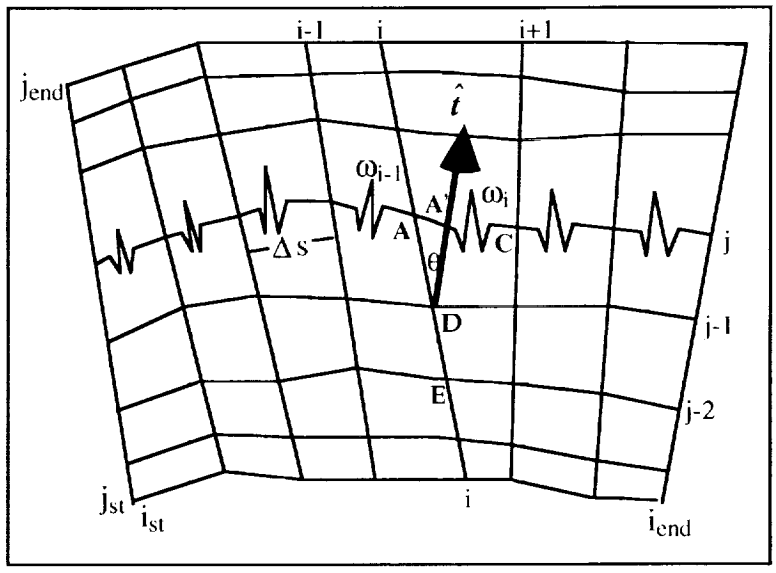

Figure 3. Current adaption line j on a 2-D surface.

In the SAGE code, this 1-D solution technique is used along the initial adaption line of the initial adaption plane, where no directional information is available. Continuing this approach for successive line-by-line adaptions will create a mesh that is insufficiently smooth for input into computational flow-field codes. Therefore, to ensure a more reasonable grid, the redistribution of points (driven by the tension springs) is constrained by torsion terms defined on two adjacent adapted lines: one on the current plane and one on the preceding plane. Within the current plane, the torsion parameter τ represents the magnitude of the torsion force that maintains smoothness and orthogonality between the node *(i,j,k)* and the nodes *(i,j–1,k)* and *(i,j–2,k)*. This is the only torsion parameter available on the initial surface. For subsequent surfaces, the torsion parameter

ψ constrains the movement between the same node *(i,j,k)* and the nodes *(i,j,k–1)* and *(i,j,k–2)* on adjacent computational planes. The torsion terms are evaluated as $\tau_i(s_i'-s_i)$ and $\psi_i(s_i^*-s_i)$ (see Fig. 3: s' is at A'), where the torsion parameters τ and ψ define the magnitude of the torsion forces and $s'$and s^* define their inclination (i.e., orthogonality and smoothness).

To introduce the torsion forces to the system of equations, we first rewrite Eq. (2) to represent the force balance,

$$\omega_i(s_{i+1}-s_i)-\omega_{i-1}(s_i-s_{i-1})=0 \tag{6}$$

and then add the torsion terms to obtain

$$\omega_i(s_{i+1}-s_i)-\omega_{i-1}(s_i-s_{i-1})+\tau_i(s_i'-s_i)+\psi_i(s_i^*-s_i)=0 \tag{7}$$

which is rearranged to produce the coefficients of the tridiagonal matrix used to solve for s_i,

$$\omega_{i-1}s_{i-1}-(\omega_i+\omega_{i-1}+\tau_i+\psi_i)s_i+\omega_i s_{i+1}=-\tau_i s'-\psi_i s_i^* \tag{8}$$

This equation is written for each interval along the adaption line, producing a system of (n_i-1) equations. Since s_1 and s_n are known, there are n_i-2 rows in the matrix. This equation is solved iteratively (updating ω_i at new s_i) until $\sum_{i=1}^{n_i}\left|s_i^{(n)}-s_i^{(n-1)}\right|<10^{-3}\times s_{max}$. This system of equations is central to the adaption technique used in the SAGE code and most of the description that follows pertains to deriving the coefficients of this equation.

1.3 Auxiliary Equations

1.3.1 Tension parameter, ω_i. As described in the formulation, ω_i acts as a tension force in the interval $(s_{i+1}-s_i)$ and can be imagined to be a spring (aligned with the grid line) connecting the two grid points, as shown in Fig. 3. This tension parameter is defined in Eq. (3) as

$$\omega_i = 1+A\bar{f}_i^B$$

where $\bar{f}_i$ is a function of the gradient of the flow variable, q:

$$\bar{f}_i=\frac{f_i-f_{min}}{f_{max}-f_{min}} \text{ and } f_i=\frac{\partial q_i}{\partial s}=\frac{q_{i+1}-q_i}{\Delta s_i} \tag{9}$$

The standard format of the user's input flow-field file contains the conservative flow variables Q, as defined in the plotting software package PLOT3D (Walatka et al., 1990). This format (for a 3-D dataset) assumes the flow variables are ρ, ρu, ρv, ρw, and e. Since the adaption is based on a scalar function q, this function can be evaluated as a user-specified combination of flow variables Q. Pressure, Mach number and temperature ratio are also provided as adaption variables and are internally computed, assuming the perfect gas relationship. However, Q need not be restricted to conservative flow variables, but can be any combination of user-specified flow variables that represent the flow field.

To remove the unwanted oscillations from the discrete evaluation, f_i is smoothed by adding a second derivative term, i.e., $f_i=.75f_i+.125(f_{i+1}-f_{i-1})$. By default, two smoothing passes are made, but this can be overridden by changing the filter input control parameter, NFILT. The tension parameter, ω_i, is smoothed in the same way.

1.3.1.1 *A* and *B*. As seen in Eq. (3), ω is also a function of A and B. These variables are the important elements of the self-adaptive nature of the scheme because they control the size limits of the grid spacing. A and B are computed from the user-supplied maximum and minimum allowable grid spacings, Δs_{MAX} and Δs_{MIN}. These are relative values of mesh size and allow the user to specify the proportion of largest to smallest Δs in the final adapted grid.

The value of A is constant throughout the grid adaption and is given by

$$A=\frac{\Delta s_{MAX}}{\Delta s_{MIN}}-1 \tag{10}$$

The value of B is computed (by an iterative process) for each j line to provide

$$input\,\Delta s_{MIN} = computed\,\Delta s_{min}$$

That is, the computed minimum grid spacing is equal to the user-requested value. Appendix I gives a detailed description of the derivation of these two parameters.

1.3.1.2 Torsional effect on the evaluation of ω. As just noted, the function of variables A and B is to control the size limits of the adapted-grid mesh spacing and, as described in Appendix I, B is computed from the 1-D Eq. (2). Thus the addition of the torsion terms is somewhat inconsistent with the value of B, and the resulting grid spacings may give values slightly outside the requested limits. To provide for additional control, a weighting factor is applied to ω such that $\omega_i = (1 + A\bar{f}_i^B)\omega_{t_i}$, where ω_{t_i} is the weighted tightness factor. When equation (8) is solved, each Δs_i is tested and if it does not lie within the requested spacing limit, ω_{t_i} will be computed; otherwise $\omega_{t_i} = 1.0$. If $\Delta s_i < \Delta s_{MIN}$, we need to relax (decrease) the value of ω_i to produce a larger Δs_i in the next iteration. If Δs_i is too large, ω_i should be increased. We therefore use the modifier

$$\omega_{t_i} = \frac{1}{2}\left(\frac{\Delta s_i}{\Delta s_{MIN}} + 1\right) \text{ or } \omega_{t_i} = \frac{1}{2}\left(\frac{\Delta s_i}{\Delta s_{MAX}} + 1\right)$$

depending on whether $\Delta s_i < \Delta s_{MIN}$ or $\Delta s_i > \Delta s_{MAX}$. Equation (8) is therefore solved using the modified values of ω. It should be noted that $\omega_{t_i} = 1.0$ in the regions of side-spacing control (see section 1.4.3) where Δs_i is permitted to be outside the specified limits.

1.3.2 Torsion terms. As shown in Eq. (7), the torsion terms, $\tau_i(s_i' - s_i)$ and $\psi_i(s_i^* - s_i)$, are added to the 1-D equation to preserve the required smoothness and orthogonality of the grid. They are constructed as being analogous to the behavior of torsion springs. Consider first the torsion term within the plane, $f(\tau)$. A torsion force, T, relates the node at $(i,j-1,k)$ to the node (i,j,k), is proportional to the turning angle, θ, shown in Fig. 3, and is defined as

$$T_i = \kappa\theta_{i,j-1,k} \tag{11}$$

where κ is a torsion-related constant. The next step is to relate this torsional force to the previously defined parameters on the adaption line. Since θ is generally small, we can approximate θ_i by $(s' - s_i)/|DA'|$ (see Fig. 3), where $(s' - s_i)$ is a function of the inclination angle computed from the proportioned orthogonality and straightness vectors, as described in section 1.3.2.3. In addition, we assume that κ is proportional to the maximum ω_i $(= 1 + A)$ and the local aspect ratio of the grid cell. Hence we can write

$$\kappa = \frac{\lambda\omega_{max}(s_{i+1,j-1} - s_{i-1,j-1})}{2|DA'|} \tag{12}$$

where λ is the proportionality coefficient and is an input quantity. The variable τ used in Eq. (8) can thus be defined as

$$\tau_i = \frac{\lambda\omega_{max}(s_{i+1,j-1} - s_{i-1,j-1})}{2|DA'|^2} \tag{13}$$

The evaluation of ψ is similar, with the node $(i,j-1,k)$ replaced by $(i,j,k-1)$ and a proportionality coefficient λ^* defined that is analogous to λ such that:

$$\psi_i = \frac{\lambda^*\omega_{max}(s_{i+1,j,k-1} - s_{i-1,j,k-1})}{2|BA'|^2} \tag{14}$$

where $|BA'|$ is the distance from *(i,j,k–1)* to the location of $s^*_{i,j,k}$. The evaluations of $|DA'|$ and $|BA'|$ are given in Appendix II.

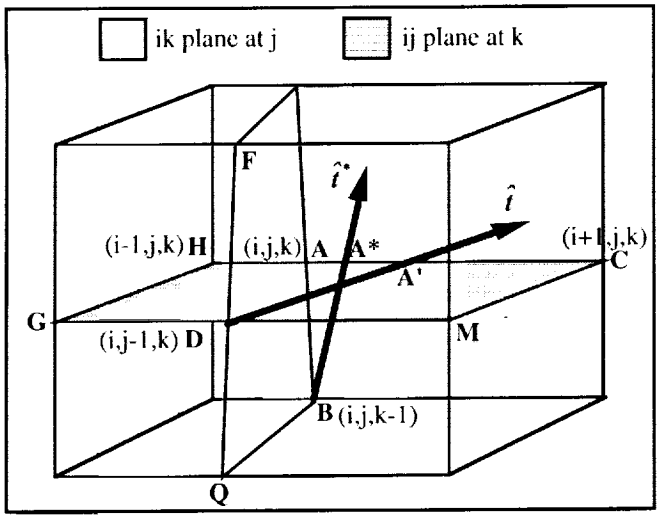

Figure 4. Torsion vectors, $\hat{t}$ and $\hat{t}^*$.

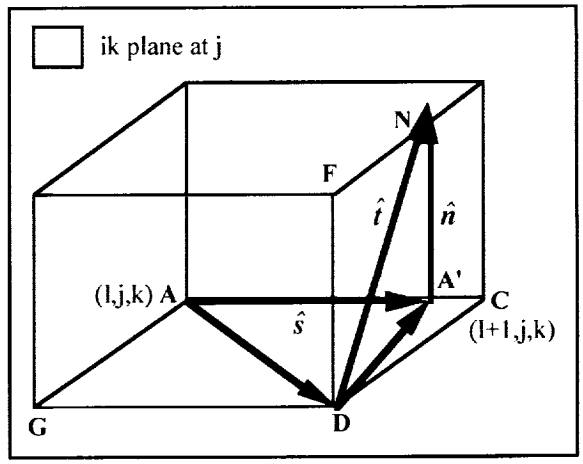

Figure 5. Calculation of A', intersection of torsion and streamwise vectors.

1.3.2.1 Evaluation of s' and s^*. These two variables, seen on the right-hand side of Eq. (8), are calculated from the torsion vectors and provide the direction of the smoothness and orthogonality constraints. The location of s' lies at the intersection of the projection of the within-plane torsion vector $\bar{t}_i$ with the arc-length vector $\bar{s}_i$ along the current adaption line. The location of s^*_i is similarly determined from the between-plane torsion vector $\hat{t}^*_i$. Figure 4 shows the two unit torsion vectors: $\hat{t}$, acting from point D and intersecting the line AC at A', and $\hat{t}^*$, acting from point B and intersecting the same line at A^*, such that

$$s'_i = s_i + AA' \quad \text{and} \quad s^*_i = s_i + AA^* \tag{15}$$

In general, neither $\bar{t}$ nor $\bar{t}^*$ will directly intersect the line AC, and the projection of the torsion vectors onto $\bar{s}$ is required. An example of the projection vectors is shown in Fig. 5 where the projection of $\bar{t}$ onto the k plane intersects $\bar{s}$ at A'. A full description of the calculation of the intersection is given in Appendix II.

1.3.2.2 Torsion vectors $\bar{t}$ and $\bar{t}^*$. Figure 6 shows in detail the within-plane torsion vector $\hat{t}$, and the associated base vectors ($\hat{n} = f(\hat{u}, \hat{b})$, and $\hat{e}$). The vectors in these figures are displayed on a 2-D surface, and it should be remembered that the shown $\hat{t}$ is actually the projection of $\hat{t}$ seen in Fig. 5. The unit vector $\hat{t}$, associated with the nodal point *(i,j,k)* [but acting from the point *(i,j–1,k)*] is defined as

$$\hat{t}_{i,j,k} = \frac{1}{|t|}[C_t \hat{e}_i + (1 - C_t)\hat{n}_i] \tag{16}$$

where

C_t is the input parameter that defines the percentage of straightness to orthogonality,

$\hat{e}_i$ is an average straightness vector from *(i,j–2,k)* to *(i,j–1,k)*, and

$\hat{n}_i$ represents the orthogonality vector between the *j–1* line and the node *(i,j,k)* and is a function of $\hat{b}$ and $\hat{u}$ vectors described below.

The between-plane torsion vector $\bar{t}^*$ which acts from the node *(i,j,k–1)*, is similarly defined as

$$\hat{t}^*_{i,j,k} = \frac{1}{|t^*|}[C^*_t \hat{e}^*_i + (1 - C^*_t)\hat{n}^*_i] \tag{17}$$

where C_i^* is the input proportion parameter, and $\hat{e}_i^*$ and $\hat{n}_i^*$ are the equivalent straightness and normal vectors.

It should be noted that Figs. 4 and 6 show vectors intersecting the j line in the interval $(i,i+1)$; however, this is not necessarily the case. In general, we assume that the projection of $\hat{i}_i$ intersects the j line in the interval $(l,l+1)$. Appendix II describes a general method used to compute the intersection of a vector with the vector $\bar{s}$.

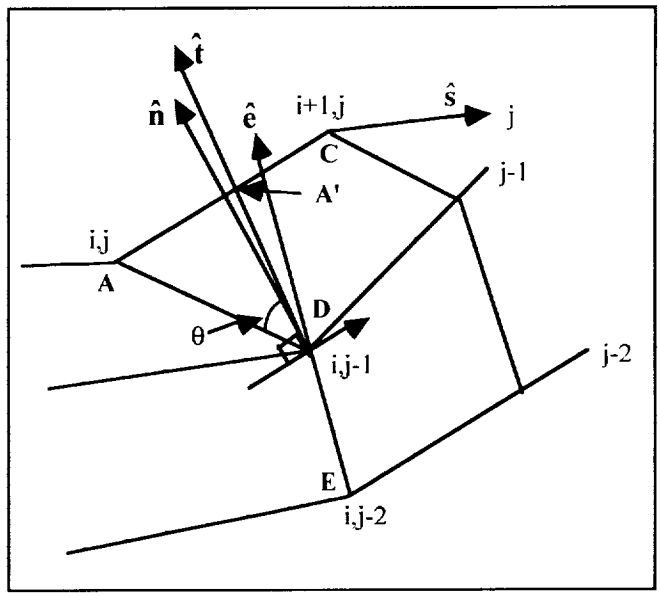

Figure 6. Torsion vector as a combination of normal and straightness vectors.

1.3.2.3 Evaluation of vectors $\hat{s}$, $\hat{e}$, and $\hat{n}$. The unit vector $\hat{s}_i$ (direction AC in Fig. 7) is the arc-length vector from s_i to s_{i+1} along a j line. It is defined as

$$\hat{s}_{i,j} = s_{x_i}\hat{i} + s_{y_i}\hat{j} + s_{z_i}\hat{k}$$

where

$$s_{x_i} = \frac{1}{|s_i|}(x_{i+1,j,k} - x_{i,j,k}),\ s_{y_i} = \frac{1}{|s_i|}(y_{i+1,j,k} - y_{i,j,k})$$

$$\text{and } s_{z_i} = \frac{1}{|s_i|}(z_{i+1,j,k} - z_{i,j,k})$$

The unit vector $\hat{e}_i$ (direction ED in Fig. 7) is defined as the sum of three straightness vectors, $\hat{d}_{i-1}$, $\hat{d}_i$, and $\hat{d}_{i+1}$, where

$$\hat{d}_{i,j} = d_{x_i}\hat{i} + d_{y_i}\hat{j} + d_{z_i}\hat{k}$$

such that

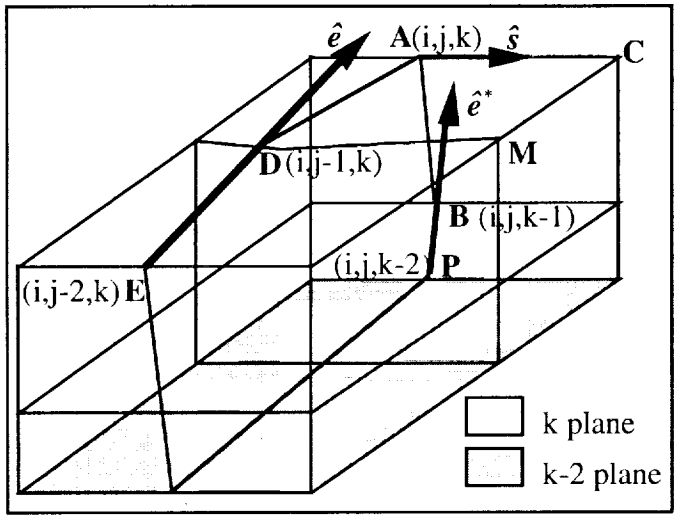

Figure 7. Straightness vectors, $\hat{e}$ and $\hat{e}^*$.

$$d_{x_i} = \frac{1}{|d_i|}(x_{i,j-1,k} - x_{i,j-2,k}),\ d_{y_i} = \frac{1}{|d_i|}(y_{i,j-1,k} - y_{i,j-2,k})$$

$$\text{and } d_{z_i} = \frac{1}{|d_i|}(z_{i,j-1,k} - z_{i,j-2,k})$$

Therefore, $\hat{e}_i = \frac{1}{\sqrt{3}}(\hat{d}_{i-1} + \hat{d}_i + \hat{d}_{i+1})$. If the line $j-2$ does not exist (such as at the second line from a physical boundary), it is assumed that $\hat{e} = \hat{n}$.

The straightness vector between planes, $\hat{e}_i^*$, is shown as $\vec{PB}$ in Fig. 7. It is computed like $\hat{e}$ above, with points $(i,j-1,k)$ and $(i,j-2,k)$ replaced with $(i,j,k-1)$ and $(i,j,k-2)$ respectively.

The vector $\hat{n}$ is a combination of two vectors:

$$\hat{n} = \frac{1}{|n_i|}\left[t_n\hat{b} + (1-t_n)\hat{u}\right] \qquad (18)$$

where $\hat{u}_i$ represents orthogonality to the ik plane through the $j-1$ line, and $\hat{b}_i$ the orthogonality to the ik plane through the j line, as shown in Fig. 8. The parameter t_n that proportions $\hat{b}$ and $\hat{u}$ is defaulted to .5, changing only at the upper and lower marching boundaries, as discussed in section 1.4.4. The calculation of

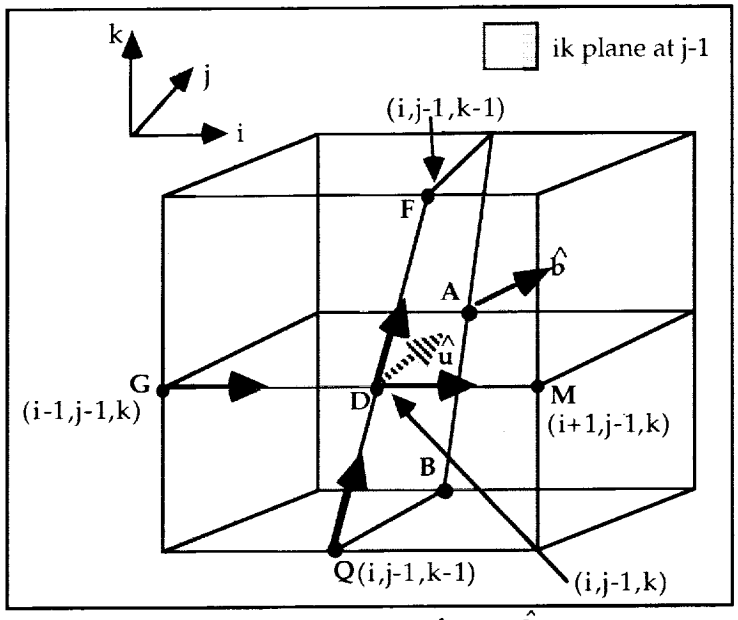

Figure 8. Normal vectors $\hat{u}$ and $\hat{b}$.

a normal vector to a point in a 3-D surface needs defining: for example, in Fig. 8, the vector $\hat{u}$ acts at *D*, perpendicular to the *ik* surface at *j–1* (the surface that contains *G, F, M,* and *Q*). In this case, there are four possible planes to choose from (*MDF, MDQ, GDF,* and *GDQ*) and thus the definition of the $\hat{u}$ and $\hat{b}$ vectors is the average of the normals to each of the four planes, when the planes exist. Since two vectors define the plane in which they lie, the normal to that plane is obtained by taking their cross product. The vector $\hat{b}$ would also act at D but is perpendicular to the *ik* plane at *j*. Appendix III describes the general calculation of the normal to a given plane at a given point.

In addition to the within-plane orthogonal constraints, orthogonality is also controlled between planes by a function of the vector $\hat{n}^*$, defined as

$$\hat{n}^* = \frac{1}{\left|n_i^*\right|}\left[t_n^*\hat{b}^* + (1-t_n^*)\hat{u}^*\right] \tag{19}$$

The normal $\hat{u}^*$ acts at point B, perpendicular to the *ij* plane at *k–1*, and $\hat{b}^*$ acts at *A*, but perpendicular to the plane at *k*.

1.4 Treatment of Boundaries

The ability to modify the adaption techniques in boundary regions substantially improves the flexibility of the adaptive scheme. The adaption domain is defined by the user and may be equal to, or a subset of, the physical grid. A boundary occurs at the limits of the adaption domain defined by the user. Within a plane, there are two types of boundaries: marching boundaries (all points along the initial and final adaption lines) and edge boundaries (at $i = i_{st}$ and $i = i_{end}$). There are also initial and final surface boundaries to take into consideration. Figure 9 shows a 2-D example of an adaption domain as a subset of the physical grid, and illustrates the two types of boundary lines when marching in the *j* direction. Note that if marching had been in the *i* direction, the boundary definitions would have been reversed.

By internally amending the previously defined variables of C_t, t_n, and λ in the boundary regions, it is possible to maintain physical characteristics at boundaries, allow for multiple passes, provide continuity across grid boundaries, and simplify multiple grid adaptions. Specialized boundary conditions, such as wall shape preservation and periodic boundaries, are discussed in separate sections.

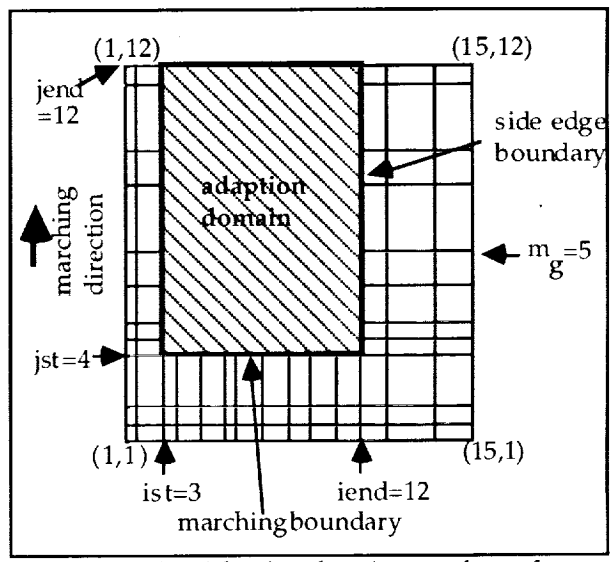

Figure 9. Adaption domain as subset of physical domain.

1.4.1 Treatment of initial marching boundary line. If the initial adaption line is within the physical domain, a smooth transition will be

required across the starting internal boundary. For example, this situation occurs when adapting in zones; each zone has different flow features and the user may wish to march up to a certain line using one set of parameters, then continue marching using a new set. The common boundary between the two zones must remain unchanged when starting the second adaption pass. This feature is controlled by the input parameter m_g. When $m_g > 0$, a smooth transition from external grid lines (e.g., those in the already adapted zone) to internal lines is created by maintaining the same grid points along the initial line and then incrementally introducing the input adaption parameters. For example, when stepping to the second adaption line, most straightness is maintained by setting $C_t = 1$ (see Eq. (16)). At each subsequent line, C_t is gradually decreased until it equals the user-supplied input value. The number of lines stepped before the full effect of the new parameters is felt depends on m_g; C_t will linearly decrease until m_g lines have been adapted. An example of this can be seen in Fig. 9, in which $j_{st} = 4$ and $m_g = 5$. The grid points along $j = 4$ will remain unchanged while those along $j = 8$ will be fully adapted to the input control parameters. The code controls the adaption parameters for lines in between. Consequently, we define a variable n_m as

$$n_m = \frac{m_g - j}{m_g - j_{st}} \tag{20}$$

and replace the value of C_t used in Eq. (16) by

$$C_{t_m} = C_t(1 - n_m) + n_m$$

At the same time, the value of λ is increased to $\lambda(1 + 5n_m)$ so that this amendment to C_t is more effective. After m_g lines, $n_m = 0$, thus returning C_t and λ to their original values. Note that the computation of $\hat{e}$ along the $j_{st} + 1$ line is a function of the $j_{st} - 1$ line (if it exists), and this will also help in the merging process.

A similar parameter, m_g^*, is used when the first plane must remain unchanged (e.g., at a multigrid boundary) and subsequent planes are gradually adapted.

1.4.2 Treatment of final marching boundary line. Another discontinuity will occur between the final adaption line and any subsequent lines external to the adaption domain. Since the adaption process is a marching scheme, it is not possible to use the same merging concept described earlier. On request (through the MARCH input parameter), the grid points on the remaining external lines will be redistributed with the same proportions as the points on the final adapted line. In addition, nonadapted planes can be proportioned with respect to lines on previously adapted planes. This is not an adaption to the flow field, but provides a more acceptable interface between the computational and physical domains.

1.4.3 Treatment of side-edge boundary. A smooth transition will also be needed at the side-edge boundaries if the adaption domain is internal to the given grid, i.e., if $i_{st} > 1$ or $i_{end} < imax$. Figure 10(a) shows an example in which the fixed external grid spacing is denser than the first redistributed points, giving a discontinuous effect across the boundary. If the user requests continuity of mesh spacing across the side-edge boundaries (by setting the input parameter NEDGE, $(n_g) \neq 0$), a modification is made to the tension parameter, ω. Consider the start-edge boundary at $i = i_{st}$ along line j. We wish to enforce some value on ω_1 that will give a value of Δs_1 close to the value of $\Delta s_{i_{st}-1}$. To do this, we find the average $\omega\Delta s$ along the converged $j-1$ line and replace ω_1 by $\overline{\omega\Delta s}/\Delta s_{i_{st}-1,j}$. This value remains fixed during the iteration, but is merged into the updated values of ω close to the boundary. In general, this implies that ω_2, ω_3, and ω_4 will be amended; however,

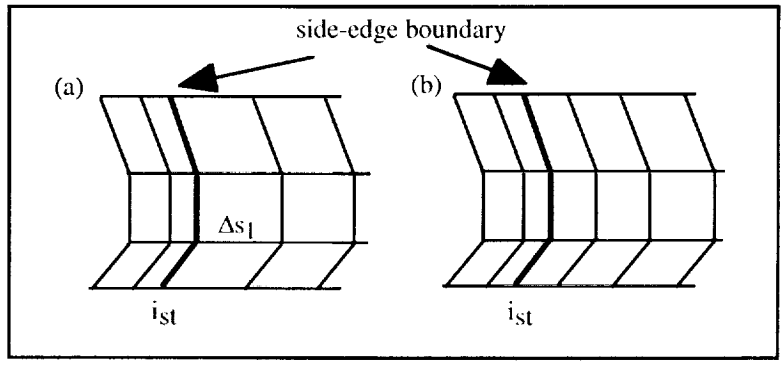

Figure 10. Control of side-edge boundaries. (a) With no edge control; (b) with edge control.

an additional option is available (using input parameters MG1 and MG2) to spread the effect further into the adaption domain. Figure 10(b) shows the effect of this process, giving a more appropriate spacing in the vicinity of the i_{s_l} boundary. The end-edge boundary is handled in a similar manner.

The side-edge spacing often needs to be improved even when the adaption domain coincides with the physical domain. If an adaption pass generates inappropriate side-edge spacing, the code can be rerun with n_g set to nonzero in an attempt to improve the spacing by using the above technique. In this case we do not have an external Δs, and $\Delta s_{2,j}$ is used instead of $\Delta s_{i_u-1,j}$. This will usually prevent the spring constants from pulling the lines too far off the boundary.

The variable n_g can be set to initiate computation for either or both side edges.

1.4.4 Treatment of orthogonality at boundaries. The code provides the choice of constructing grid lines that are as orthogonal as possible to a marching boundary, either to the final line in a plane or to the entire final plane. To accomplish this, the normal vector $\hat{n}$ (and/or $\hat{n}^*$) is emphasized over the straightness vector by decreasing C_t when the adaption line is close to a marching boundary line. For even greater control, the coefficient t_n is modified to emphasize either $\hat{u}$ or $\hat{b}$, depending on whether the initial-line or end-line boundary is being considered. To ensure that the modifications to these torsion coefficients sufficiently affect the computation, the value of λ is simultaneously increased to accentuate the torsion term. Since not all marching boundaries are physical boundaries, an input option is available that will override this emphasis on orthogonality for either boundary.

1.5 Preservation of Initial Wall Boundary Shape

In applications for which the shape of the wall boundary is defined by large geometric gradients, such as for turbine blades, sharp corners, and the leading edges of airfoils and wings, sufficient points need to be placed in the appropriate regions of the initial grid to accurately define the geometry. If flow-field gradients are weak in these regions, the standard grid-redistribution algorithm will cause points to be dispersed, leaving insufficient points to properly describe the surface. To maintain necessary clustering in these regions, on user request a new variable is introduced that is a function of the geometry gradients that define the boundary shape. The weighting parameter now becomes a function of both flow-field gradients and geometry gradients. The solution procedure will therefore redistribute the points into regions of high geometry gradients as well as into regions of high flow-field gradients. Both the start and end wall boundaries are treated. A merging technique is used to integrate the boundary and internal redistribution, so that the internal flow is controlled only by the flow-field gradients.

The original weighting function was defined in Eq. (3) as $\omega = 1 + A\bar{f}^B$ where $\bar{f}$ is a function of $\partial q/\partial s$. When the geometry option is invoked, $\bar{f}$ becomes a function of both $\partial q/\partial s$ and $\partial g/\partial s$, where $f(\partial g/\partial s)$ is equal to the radius of curvature R at the wall boundaries. The radius of curvature is defined as

$$R = \left(\frac{\partial x}{\partial s}\frac{\partial^2 y}{\partial s^2} - \frac{\partial y}{\partial s}\frac{\partial^2 x}{\partial s^2}\right) \Bigg/ \left(\frac{\partial x^2}{\partial s} + \frac{\partial y^2}{\partial s}\right)^2$$

where the derivatives are computed from the spline coefficients.

The full effect of the geometry function should be felt at the wall boundaries; it should not be a factor in the internal grid redistribution. Hence the final weighting function takes the form

$$\omega = 1 + A\bar{f}^B$$

where $\bar{f} = C_q f(q) + C_g f(g)$ and the constant C_q normally equals 1. The constant C_g equals 1 when the upper and lower wall boundaries are being adapted, but it must be decreased away from the walls until $C_g = 0$ internally. The value of the aspect ratio was chosen to drive the rate of decrease of C_g; i.e., $C_g = 1 - 4A_s$, where A_s is the aspect ratio at the maximum radius of curvature. Regardless of the value of A_s, C_g remains positive for a minimum of four steps from the boundaries and decreases smoothly. When the upper boundary is approached, C_g must be turned on when necessary and increased with each step, and $f(g)$ becomes a function of the upper wall boundary only.

It is also possible to adapt only to the geometry gradient. If this is requested, $C_q = 0$ and $C_g = 1$ throughout the flow, and $f(g)$ is computed for each adaption line, based on the local geometry of that line. This option has proven to be a useful tool for improving the starting grid before any solution is obtained. Points will be smoothly clustered with respect to the geometry gradients.

1.6 Finite-Volume Grids

In applications using finite-volume techniques, each flow-field variable (q) is evaluated at the cell center instead of the nodal point of the grid. These cell centers are usually positioned at the average of the surrounding grid points (four for 2-D and eight for 3-D). This creates a solution file that has one less point in each direction than the grid file. The finite volume module in SAGE interpolates the q values onto the grid points before performing the adaption. On conclusion the q values are re-interpolated at the new cell centers.

A feature of finite volume grids is the definition of the outermost boundary cells in all computational directions as 'ghost cells' containing specified flow values. The adaption procedure destroys these values and SAGE employs a very simple method to replace them: on output, each ghost cell contains the same value of q as the adjacent internal cell. It is therefore highly recommended that the user re-apply the boundary conditions before processing the new grid. This is true for both 2-D and 3-D grids and implies that if a user wishes to adapt one plane for testing purposes, an internal plane would be more appropriate.

1.7 Multiple Grids

When investigating the flow around complex structures, computational grids are frequently organized in multiple-grid format. This enables each individual grid to be of manageable size while maintaining a single dataset and also allows for overlapping grids. The original SAGE code could only adapt single grids and it was therefore necessary to separate the grids before adaption. The latest version of SAGE can read and write datasets in multigrid format and provides an input control parameter to specify which grid to adapt. In addition, a feature is available to transfer data between matching zonal boundaries (with 1:1 mapping) in separate grids. Some or all boundaries of each individual grid will match in some part to boundaries in one or more of the other grids. It is important that common boundaries, where data in separate grids represent the same location in computational space, retain the same grid and flow distributions. This can be seen in the simple two-part multiple grid shown in Fig. 11(a), where the shaded plane is common to both grids. After adaption, both planes must still contain matching data.

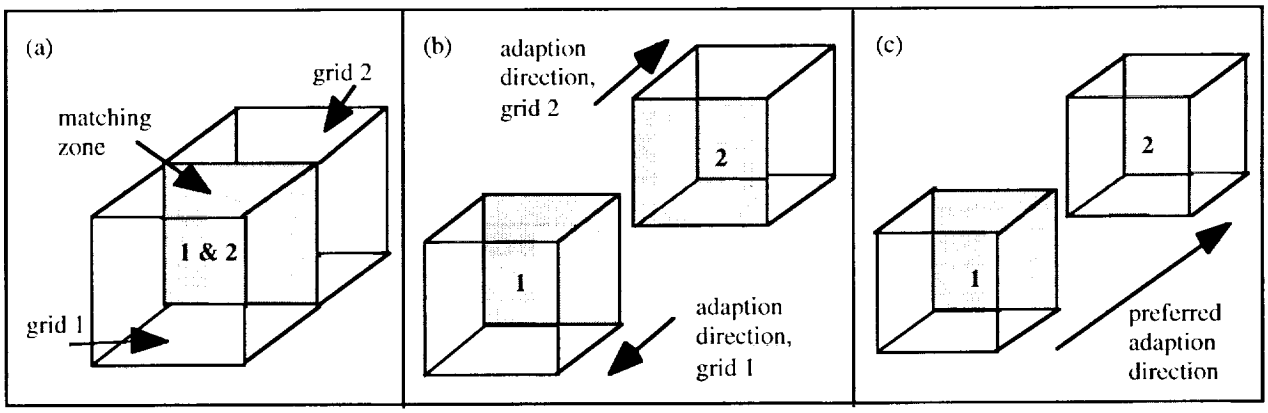

Figure 11. Simple multi-zone grid. (a) Common plane; (b) restricted adaption direction with original SAGE; (c) preferred adaption direction, using plane transfer procedure.

The adaption technique used in SAGE is a marching scheme, and therefore the order and direction of adaption have a marked effect on the final grid-point redistribution. One way to adapt the two grids and still retain the common boundary data is shown in Fig. 11(b): the common plane is adapted first, with plane marching occurring in opposite directions. The *export/import* feature enables the order of adaption shown in Fig. 11(c): after grid 1 has been adapted, adapted data from the final plane can be transferred to the first plane in grid 2. Then, using the merging feature (MGPLS) to prevent the first plane of grid 2 from being

corrupted, adaption can continue in the same plane marching direction. All three steps (adaption of grid 1, data transfer, adaption of grid 2) can be accomplished in the same execution pass. Figure 11(c) shows only the simplest multigrid structure. In reality, multiple grids may contain many grids, with surfaces, or subsets of surfaces, matching to more than one grid. Input options are available to handle this and examples can be found in Section 3.

1.8 Outer boundary movement

1.8.1 Moving the outer boundary. Once a solution has been obtained on an initial grid, it may be apparent that the location of the outer boundary of the grid is inappropriate. For example, a shock wave may have developed that passes out of the boundary; or alternatively, no features can be seen in part of the outer region and much of the grid is wasted. The ability to move the outer boundary is a very useful feature and is invoked by adding the input option MVBOUND to the input parameter list. The new boundary location moves along each existing grid line. If the boundary moves out, the new location lies on the extension of the vector joining the last two existing points. This feature has been extensively used for applications that compute flow solutions on a vehicle configuration flying a trajectory over a wide range of angles of attack.

The user has three options for determining the new boundary. The first is the most simple: moving the boundary point a certain percentage from its current location, either in or out. Along each adaption line, the value of s_{max} is increased or decreased by a user-supplied percentage (controlled by MVBOUND=$\pm p$, where p is the percentage), and then transformed back to (x,y,z) to form a new boundary point.

The second option permits the user to parallel the outer shock (MVBOUND=999.) defined by the requested variable, INDQ. The outer boundary will be moved in or out for each adaption line, depending on the location of the shock. A search is made, starting at the outer boundary point of each grid line, and moving towards the inner boundary. As soon as a significant change in gradient is encountered [given as $f(q_i) - f(q_{i-1}) > .0001$ where $f(q_i) = \frac{\partial q_i}{\partial s} \Big/ \left.\frac{\partial q}{\partial s}\right|_{max}$], it is assumed that the shock location, s_{ishock}, has been found. The relative distance of the new boundary point from the shock location is a function of d_{sn}, also provided by the user, such that $s_{max_{new}} = s_{ishock} + s_{max_{old}} * d_{sn}/100.0$. If a shock is moving out from the current boundary, this method will only expand the grid by a given amount and may not project the shock location sufficiently outside the boundary. If it is obvious that the shock will still not be contained within the new boundary, moving the boundary out with option 1 and then applying this second option, will be more appropriate. Note that when the new location is dependent on a gradient location, the outer boundary may not be smooth and the smoothing parameters NSM and NSM1 may be required. The discontinuities occur since the outer regions of a grid (where a shock may be found) are often the most sparse.

The final option is to move the boundary location based on a constant specified value of a flow variable, for example, a line parallel to a certain density contour. Note that only variables given in the input Q file can be used to move a boundary, not any internally computed values.

1.8.2 Point redistribution after boundary movement. Once the new boundary edge has been located and the new (x,y,z) at s_{max} computed, the internal grid points must be redistributed. Adaption is not an option: the code will automatically set NOUP=.true. A smoother point distribution is first required, then adaption can take place as a subsequent step.

There are two options for reclustering points without adaption to the flow. With no other information, the code will automatically redistribute the points proportional to the original grid-point distribution, i.e.,

$$s_{i_{new}} = s_{i_{old}} s_{max_{new}} / s_{max_{old}}$$

The second reclustering method is based on the stretching algorithm of Vinokur (1983) and is invoked by inputting RECLUST>0 and providing information for the first and last mesh spacing along the reclustering line.

1.9 Vinokur Reclustering

This method is appropriate for redistributing grid points when very small mesh spacings are required at the wall as, for example, in viscous grids. The user can specify the first spacing off the wall and this algorithm will create a smooth mesh-size expansion through the boundary layer. The method can be used to redistribute points after a boundary movement, or as a stand-alone alternative to the basic grid adaption procedure.

The reclustering method is based on the 1-D stretching algorithm of Vinokur (1983) and is invoked by inputting RECLUST>0 and providing information for the first and last mesh spacing along the reclustering line. The redistribution can apply to the entire domain (RECLUST=*imax*) or to a portion of the domain (e.g., RECLUST=*n* will redistribute the first *n* points from the wall).

The Vinokur algorithm redistributes points as $s_{i_{new}} = f(d_w, d_e)$ where d_w and d_e are user input variables that help to define the first and last Δs spacings needed by the algorithm. The last Δs spacing, Δs_{vmax}, is calculated as

$$\Delta s_{v\,max} = d_e s_{max}/(npts-1)$$

where the default value of d_e is 5.

There are three options to assign a value for the first wall spacing, Δs_{wall}:

1) default to the original Δs_1;
2) input d_W as the required value of Δs_{wall}; or
3) input $d_W = 999$ and then input the first (*j=1*) and last (*j=jmax*) wall spacings.

Option 3 is provided to overcome a 1-D limitation of the algorithm: that there is no smoothness from line to line. This is emphasized in regions where the value of Δs_{wall} is too small with respect to s_{max}. If the grid is expanding in size downstream, the Δs_{wall} that is appropriate near the vehicle nose may be too small at the outflow plane. Option 3 is provided to vary the first spacing off the wall while marching along the body surface. If $d_w = 999$, then d_{w_1} and d_{w_2} must be input. These are the wall spacings at the first line *(j=1)*, and last line *(j=jmax)* and enables d_w to be computed internally for each *j*. If p_j is the distance along the wall (at *i=1*) from the current *j* line to the *j=1* line, then

$$\Delta s_{1_j} = a + b\sqrt{p_j} \text{ where } a = d_{w_1} \text{ and } b = (d_{w_2} - d_{w_1})/\sqrt{p_{j\,max}}$$

As mentioned above, it is possible to recluster the grid points without adapting and without moving the outer boundary. An input of RECLUST=*n*, *n>3* and MVBOUND=0 will invoke the re-clustering algorithm between points s_1 and s_n. Here, *n* could indicate the edge of the boundary layer, or the entire grid. If *n<imax*, the 'outer' edge spacing sent to the reclustering algorithm is $\Delta s_n = (s_{n+1} - s_{n-1})/2$.

1.10 Optional Versions of SAGE

1.10.1 Blanked grids. Certain types of complex multiple-grid structures utilize the facility in PLOT3D that blanks out regions of overlapping grids. However, the use of multiple, overset grids with blanking creates a new environment for the adaption algorithm. The adaption procedure used in the SAGE code is a 1-D approach; i.e., a single line is adapted before stepping to the adjacent line. The integrity of the adapted grid is maintained by defining restraining forces between the current adaption line and previously adapted lines, either within the current plane or on the previously adapted adjacent plane. The method assumes that there are an equal number of grid points along each line; i.e., there is a one-to-one correspondence between a point and its associated restraining forces.

Overset grids are created in the normal structured manner, with equal points along each line. However, a flag associated with each grid point indicates whether the point should be included as an active part of the grid. This method creates adjacent grid lines with different numbers of active grid points, and even more problematical, regions of grid holes (i.e., one or more gaps along the line). The challenge for the SAGE algorithm and code is to develop a method that can accommodate these complications. Section 5 of this document contains a description of the new method that was developed and it can be seen that it adds complexity to the code. As a result, it was decided not to include this feature in SAGEv3 at this time. However, SAGEv2B (Version 2 with Blanking) is available from the authors. The description in Section 5 applies to this code but will remain appropriate if the feature is added to a later version of SAGE.

1.10.2 Cyclical and periodic boundaries. Certain classes of grids used by flow-field solvers are not suitable for adaption using the basic grid formulation described thus far. The self-adaptive grid procedure is a marching scheme; i.e., the solution along each line is influenced only by previously adapted lines. The adaption of the first grid line is based on flow gradients only, but as marching proceeds, the redistribution of points is dampened by the torsion effect. As a result, the final adaption line will be somewhat less adapted to the flow-field gradient than the initial line. This implies that the scheme cannot be directly applied to cyclical and periodic grid structures, since common and/or matching boundaries will show as discontinuities. For 2-D problems, these difficulties have been overcome by including both an iterative and a rearrangement procedure that can be requested by the user. However, these options are not provided in this version of SAGE even when TWOD is specified, and interested users should contact the authors to obtain a copy of SAGE2D and its documentation. For 3-D grid structures, maintaining periodicity at matching planes is a more difficult task. Since the initial grid and flow field on matching first and last planes are identical, setting the plane torsion parameter (λ^*) to zero produces matching adapted grids on these planes. In this case, the adaption of intermediary planes has no influence on subsequent planes.

1.11 Appendices

1.11.1 Appendix I: Derivation of *A* and *B*. A and B provide self-adaptiveness to the adaption scheme. These two parameters ensure that all repositioned grid nodes maintain grid spacing to within the user-requested mesh-size limits (Δs_{MIN} and Δs_{MAX}).

1.11.1.1 Calculation of *A*. We wish to relate A to the input values of Δs_{MIN} and Δs_{MAX}. A is constant throughout the entire mesh, and hence the 1-D relationship given in Eq. (2) holds. From the original definition of $\bar{f}$ in Eq. (9), $\bar{f}_{max} = 1$ and $\bar{f}_{min} = 0$; therefore, from Eq. (3) we have $\omega_{max} = 1 + A$ and $\omega_{min} = 1$. From Eq. (2) (rewritten as $\Delta s_i = K/\omega_i$) we can see that the minimum Δs will occur at K/ω_{max}. Similarly, the maximum Δs occurs at K/ω_{min}. We therefore wish to set $\Delta s_{MIN} = K/(1+A)$ and $\Delta s_{MAX} = K$. These can be solved simultaneously: by eliminating K we get the expression given in Eq. (10)

$$A = \frac{\Delta s_{MAX}}{\Delta s_{MIN}} - 1$$

1.11.1.2 Calculation of *B*. This calculation is more complex. B is found by an iterative procedure and will change for each j line. The objective is to determine which value of B will give the minimum computed Δs_i equal to the requested minimum, Δs_{MIN}. For each value of B there exists a minimum Δs_i. (An example of a plot of B vs. Δs_{min} is shown in Fig. 12). We need to find B at the requested Δs_{MIN}. To do this, we assume an initial value of $B(=1.0)$, evaluate ω, and then solve for the new Δs using Eq. (5). Although Eq. (5) is true only for the initial line, it is assumed here to hold for all j in order to simplify the B calculation. If $\left|\Delta s_{MIN} - min\Delta s_i^{(n)}\right|$ is small, an acceptable value of B has been found. If not, a new B is computed from $B^{(n+1)} = B^{(n)} + \Delta B^{(n)}$, ω is reevaluated, and the procedure repeated.

$B^{(n)}$ can be found from the definition

$$\frac{\partial}{\partial B} min(\Delta s_i) = \lim_{\partial B \to 0} \frac{(\Delta s_{MIN} - min\,\Delta s_i^{(n)})}{\Delta B^{(n)}} \qquad (21)$$

As mentioned in the calculation of A, we know that Δs_i is a minimum when $\omega_i = 1 + A$, and by substituting this in Eq. (5) and differentiating, we obtain

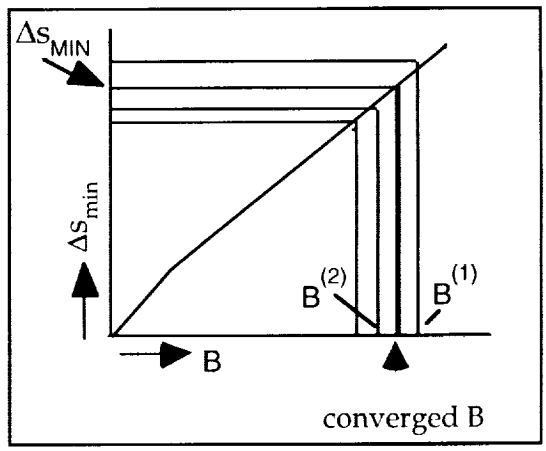

Figure 12. Calculation of B.

$$\frac{\partial}{\partial B} min(\Delta s_i) = \frac{s_{max}}{1+A} \frac{\partial}{\partial B}\left(\frac{1}{\varphi}\right) \tag{22}$$

where

$$\varphi = \sum_{l=1}^{n_i} \frac{1}{\omega_l}$$

We know that

$$\frac{\partial}{\partial B}\left(\frac{1}{\varphi}\right) = -\frac{1}{\varphi^2}\frac{\partial \varphi}{\partial B}$$

Hence, the next step is to evaluate $\partial\varphi/\partial B$. With this definition of φ, we can take the summation sign out of the differential and obtain

$$\begin{aligned}\frac{\partial}{\partial B}\left(\sum_{l=1}^{n_i}\frac{1}{\omega_l}\right) &= \sum_{l=1}^{n_i}\frac{\partial}{\partial B}\left(\frac{1}{\omega_l}\right) \\ &= \sum_{l=1}^{n_i}\frac{\partial}{\partial B}\left(\frac{1}{1+A\bar{f}_l^{\,B}}\right) \\ &= -A\sum_{l=1}^{n_i}\frac{\bar{f}_l^{\,B}\log\bar{f}_l}{\omega_l^2}\end{aligned}$$

Equation (22) can now be solved, and after substituting for

$$\sum_{l=1}^{n_i}\left(\frac{1}{\omega_l}\right) = \frac{s_{max}}{\Delta s_{min}(1+A)}$$

from Eq. (5), we obtain

$$\frac{\partial}{\partial B} min(\Delta s_i) = \frac{A(1+A)}{s_{max}}\left[min(\Delta s_i)\right]^2 \sum_{l=1}^{n_i}\frac{\bar{f}_l^{\,B}\log\bar{f}_l}{\omega_l^2}$$

Finally ΔB can be obtained from Eq. (21), giving B.

1.11.2 Appendix II: The intersection of a 3-D vector, $\bar{v}$, with the arc-length vector, $\bar{s}$. This technique is used to obtain the $\hat{b}$ vectors used in the orthogonality terms and to find the location of s_i' and s_i^*. For example, to determine s_i' we need the segment in which the torsion vector $\hat{t}$ from the grid point *(i,j−1,k)* intersects the *j* line. To evaluate the direction cosines of the $\hat{b}$ vector, we need to find the segment $l \rightarrow l+1$ along the *j* line that contains the intersection of a vector acting from *(i,j−1,k)* that is normal to the *j* segment. For s_i^* and $\hat{b}^*$, we are concerned with the line *j* in the plane *k−1*. In all cases, the technique for finding *l* is the same.

Figure 5 shows the case of the torsion vector $\hat{t}$, computed for the point *(i,j,k)* and emanating from the point *(i,j−1,k)* at *D*. This vector $\hat{t}$ most likely will not lie in the plane described by *ADC* (or the equivalent *i* plane) and thus its projection onto the plane is required. The value of s_i' in the solution equation lies at A', the intersection of the projection of $\hat{t}$ onto the *ADC* plane and the arc-length vector, $\bar{s}$. The vector $\hat{n}$ $(= n_x\hat{i} + n_y\hat{j} + n_z\hat{k})$ is the unit normal to the plane *ADC* and is computed by the technique described in Appendix III. Both $|AA'|$and $|DA'|$ are required in the code.

From vector addition, we can write

$$|AA'|\hat{s} = \vec{AD} + |DN|\hat{t} - |NA'|\hat{n} \tag{23}$$

Since the coordinates of A and D are known, $\vec{AD}\,(=a_x\hat{i}+a_y\hat{j}+a_z\hat{k})$ can be evaluated, hence $|AA'|$, $|DN|$ and $|NA'|$ are the three unknowns. By equating coefficients of $\hat{i}$, $\hat{j}$, and $\hat{k}$ in the vector Eq. (23), we can obtain a set of three equations with three unknowns:

$$|AA'|s_x = a_x + |DN|t_x - |NA'|n_x, \quad |AA'|s_y = a_y + |DN|t_y - |NA'|n_y \quad \text{and} \quad |AA'|s_z = a_z + |DN|t_z - |NA'|n_z$$

These equations can be solved for $|AA'|$ by computing the relevant determinants.

$|DA'|$ is also found by utilizing vector addition. We have

$$\vec{DA'}\,(=|DA'|\hat{DA'}) = |AA'|\hat{s} - \vec{AD}$$

and, since the lengths of the left- and right-hand sides are equal,

$$|DA'|^2 = (|AA'|s_x - a_x)^2 + (|AA'|s_y - a_y)^2 + (|AA'|s_z - a_z)^2$$

1.11.3 Appendix III: Definition of normal vectors. The vector normal to a plane is required for the calculation of vectors $\hat{u}$, $\hat{u}^*$, $\hat{b}$, and $\hat{b}^*$. This appendix describes the derivation of a general unit-vector normal, $\hat{n}$, at a given point *(i,j,k)* and normal to the plane *ik* passing through the constant *j–1* line. This specific orientation is analogous to $\hat{u}$. Figure 8 shows such a vector acting at D that represents the normal to the shaded surface. In the figure it can be seen that there are four possible planes passing through *(i,j–1,k)*: *GDF, FDM, MDQ,* and *QDG*. Any vector normal required in the code is calculated as the average of the normals to each of these four planes. (When the node *(i,j–1,k)* is on or close to a boundary, only one or two of these planes will exist.)

The normal to a plane is derived from the cross product of two vectors lying within the plane, with care taken to ensure the correct order of the operation (this is why the input grid should be organized as a right-handed system). In the example shown in Fig. 8, the four cross products are $\vec{DF}\times\vec{DM}$, $\vec{DF}\times\vec{GD}$, $\vec{QD}\times\vec{GD}$, and $\vec{QD}\times\vec{DM}$. The required unit normal is thus the unit vector representing the sum of these four vectors.

This picture gives an idealized view of the defined planes. In reality, some of the required lines have already been adapted and some have not. In Fig. 8, the point F could be significantly different from D, since D is an already-adapted point and F is still part of the initial grid. The code solves this problem by forming a block of data around the current j line that includes all i for lines $j-1 \Rightarrow j+1$ for planes $k-1 \Rightarrow k+1$. Lines that have not yet been adapted (e.g., *j+1* at *k* or *j–1* at *k+1*) are proportioned with respect to the s_i at j; this relocates the points for a smoother computational effect.

2. SAGE USER GUIDE

2.1 Overview

The SAGE code is based on the self-adaptive grid method developed by Nakahashi and Deiwert (1985). This guide contains information that will enable the user to run the code with little knowledge of the mathematical concepts employed in the development of the adaption technique. Included is a detailed description of the input-control parameters, along with routine descriptions and nomenclature. The code stands alone and has been run on many computer systems. Users wishing to understand and/or amend the code can find the details of the mathematical background in the first section of this document.

The first step in the code is to read two data files: one that contains the coordinates of the grid *(x,y,z)* and another that contains the corresponding flow-field variables, *q*. It is assumed that both these files are in the format (given on the next page) associated with the plotting software package PLOT3D. A single-grid, finite-difference, 3-D format is assumed unless re-specified by the user. The next step is to adapt the grid with respect to the user-supplied input-control parameters. Adaption takes place as a sequence of one-directional adaptions, with the input-control parameters determining the order of adaption. Each plane (i.e., 3-D surface) is adapted in one direction only before stepping to the next plane. Figure 1 (see page 2) shows a small section of a 3-D grid: adaption in the *i* direction (stepping in *j*) has already been performed on the first plane. The code then steps to the next plane and performs the same directional adaption on this plane. A 3-D adaption is performed by a sequence of adaptions: once the grid has been adapted in one direction, the adaption can be repeated with a new parameter-input set describing another direction. (It should be noted that different orders of the adaption direction will produce different adapted grids.) When all adaptions are complete, the code sends the redistributed grid points and the corresponding interpolated flow variables to two new data files, also in PLOT3D format.

The analysis that generates the algorithms used in the code is given in detail in the first section of this report. Briefly, the redistribution of points is controlled by parameters related to torsion and tension springs, and by the maximum and minimum allowable grid spacings. Along each grid line, *(x,y,z)* is transformed to a 1-D arc-length variable, *s*. If we assume that adaption (within a plane, *k*) steps in the *j* direction, the tension spring constants, ω, are evaluated at each point *(i,j,k)*; i.e., $\omega_i = f(s_{i,j,k})$ and are a function of the gradients of the user-chosen flow-field variables. The torsion parameters (shown in Fig. 2) $\tau_i = f(s_{i,j,k}, s_{i,j-1,k}, s_{i,j-2,k})$ and $\psi_i = f(s_{i,j,k}, s_{i,j,k-1}, s_{i,j,k-2})$ are the link between the current line and the previously adapted lines. These are the parameters that maintain the integrity of the grid by controlling straightness and orthogonality within the plane and between planes. With the evaluation of these variables, a system of $(n_i - 2)$ equations (given in Eq. 8) is developed for the current *j* line and solved as a tridiagonal system. Once the adaption is complete for the current *j* line, the code steps to the next *j* line (either forward or backward) and repeats the same procedure. At the end of the current *k* plane, the code moves to the next plane and the process is repeated.

There are eight major steps taken in the code:

1. Input of three data files: initial grid, flow-field solution, and user-control parameters
2. Initialization and reorganization of data for computational purposes
3. Adaption along the start line on the start plane with the 1-D technique
4. For each subsequent *j* line on the initial plane:
   a. computation of variables that define the torsion and tension springs (ω and τ), and hence coefficients of the tridiagonal matrix defined by Eq. (8)
   b. iteration to find new values of *s*, and hence *(x,y,z)*
5. Repetition of step 4 until all lines are complete on this plane
6. For each subsequent *k* plane, computation of the torsion spring ψ for inclusion in the coefficients of the solution Eq. (8)
7. Repetition of steps 4 through 6 until all planes have been adapted
8. Output of new grid and interpolated flow-field files in the original format

2.2 Execution of the SAGE Code

The following is an example of the command file to run the SAGE code on a UNIX system. SAGE is written in FORTRAN and is self-contained; *sage* is the executable module.

```
cp xyz.grd fort.7      ! copy grid file to unit 7
cp q.fun fort.8        ! copy solution file to unit 8
sage < sage.inp        ! run SAGE code with sage.inp containing user control parameters
cp fort.10 xyz.out     ! name the output adapted grid file
cp fort.11 q.out       ! name the interpolated function file
```

where *sage.inp* is in namelist format ($NAMEL). {Note: Some systems may need the READ(5,NAMEL,...) statement in subroutine INITIAL amended to READ(5,NML=NAMEL,...)}.

The remaining files are in PLOT3D formatted (ASCII) or unformatted format:

xyz.grd contains the initial grid points (stored as a right-handed coordinate system)
q.fun contains the flow-field variables to which the grid is to be adapted
xyz.out contains the adapted grid points
q.out contains the flow-field variables interpolated on the adapted grid

In addition, an output message file is associated with unit 6, which usually defaults to the user's output device. For other operating systems, the user must appropriately assign the six input/output files.

The following are the read statements for single-grid 3-D PLOT3D unformatted input files:

xyz.grd:

```
READ(7)    IMAX,JMAX,KMAX
READ(7)    (((X(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
           (((Y(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX),
           (((Z(I,J,K),I=1,IMAX),J=1,JMAX),K=1,KMAX)
```

q.fun:

```
READ(8)    IMAX,JMAX,KMAX
READ(8)    FSMACH,ALP,RE,TIME
READ(8)    ((((Q(I,J,K,N),I=1,IMAX),J=1,JMAX),K=1,KMAX),N=1,NDIM)
```

where

IMAX=number of points in the *i* direction of the grid file
JMAX=number of points in the *j* direction
KMAX=number of points in the *k* direction
NDIM=number of Q variables (default=5)

Note: The read statement for formatted (ASCII) files is the same except READ(7) is replaced by READ(7,*) etc.

As seen in the above format, five flow-field variables are expected in the Q file. PLOT3D preassigns ρ, ρu, ρv, ρw, and e, but since the SAGE code requests only the index of the function, any variables may be stored. Note that it is possible to handle any number of flow-field variables by changing the value of NDIM in the parameter statement at the beginning of each subroutine and recompiling. Also contained in the parameter statement are the grid dimensional variables ID, JD, KD, and IMX. These are used to define the required maximum dimension of the grid arrays. Because of the internal switching of data, these values may not coincide with IMAX, JMAX, and KMAX. If the assigned code dimensions are too small, a message will be sent to the user stating the minimum dimension requirements for the current application. If the grid is in multiple grid format, an additional record occurs at the beginning of each file stating the number of grids, and the grid dimensions are supplied in dimensioned arrays. FSMACH, ALP, RE, and TIME are not used in the code and may contain dummy values. They are part of the PLOT3D package and are displayed on the output plots.

2.3 User Input Parameters

The file *sage.inp* is the user-supplied, input-parameter control file. The grid adaption is based on the user's choice of these input parameters, which are listed and briefly described below. This is followed by a more complete explanation of each parameter. The input file *sage.inp* uses the namelist format, since

generally only a few of the input parameters need to be changed from the default value set by the code. These default values are shown in parentheses in the list given in section 2.3.1. If more than one adaption pass is to be made (for example, a two-directional adaption or multiple grids), the subsequent adaptions can be made on the adapted grid by linking up to 10 sets of namelist inputs within the same *sage.inp* file. For multiple grids, these sets can also contain multiple export/import passes.

Before describing the input parameters, some terminology needs to be clarified:

The term physical domain is used to reference the complete grid defined by the input grid file, i.e., the grid bounded by IMAX, JMAX, and KMAX. The adaption domain is the part of the grid, as defined by the input-control file, that is to be adapted, i.e., (IST,IEND), (JST,JEND), (KST,KEND). These two domains can be equivalent. The direction *i, j,* or *k* refers to the direction of the grid coordinates as defined by the order in which they are stored in the grid file. The first index (normally containing *x*) is named the *i* direction, the second index is the *j* direction, and the third index the *k* direction. This implies that if data happened to be stored as *(z,x,y)* instead of *(x,y,z)*, *i* would represent the *z* direction. Regardless of the order, the data must be stored as a right-handed coordinate system. The adaption direction is used to define the 1-D line along which adaption (redistribution of points) takes place. In the analysis and in the descriptions below, this direction is always *i* for convenience, but the user may request *i, j,* or *k*. There are two stepping directions: one within the plane, defined as *j* in the analysis, and the other in the direction of plane marching, defined as *k*. Although the analysis and descriptions in this report assume this particular order of adaption and stepping, the code makes no such a priori assumptions, since the order is controlled by the user's input-parameter file. Reference is also made to grid boundaries. Side-edge boundaries refer to the start and end *points* at the edge of the adaption line (assumed to be *i* in the analysis). These are the edges that are controlled by the NEDGE parameter. Marching boundaries are the start and end *lines* of the stepping (within the plane) direction (assumed to be *j*). Adaption close to these lines are affected by the parameters ORTHS(1), ORTHE(1), and MGSTEPS, as well as by some internal controls. Planes juxtaposed to start and end *planes* are affected by ORTHS(2), ORTHE(2), and MGPLS.

2.3.1 Parameter control file, *sage.inp.* The input parameters, with their default values (in parentheses) and short descriptions, are described below. They are not in alphabetical order, but grouped by category or affinity. Reminder: $NAMEL must begin in column 2.

$NAMEL		
FORMI	(FALSE)	set to .true. if input PLOT3D files are formatted (ASCII)
FORMO	(FALSE)	set to .true. if formatted output PLOT3D files requested
IST	(1)	first adaption line in *i* direction
IEND	(IMAX)	last adaption line in *i* direction
JST	(1)	first adaption line in *j* direction
JEND	(JMAX)	last adaption line in *j* direction
KST	(1)	first adaption line in *k* direction
KEND	(KMAX)	last adaption line in *k* direction
IJPLANE	(TRUE)	the adaption surface lies in the *(i,j)* plane, with plane stepping occurring in the *k* direction
JKPLANE	(FALSE)	the adaption surface lies in the *(j,k)* plane, with plane stepping occurring in the *i* direction
IKPLANE	(FALSE)	the adaption surface lies in the *(i,k)* plane, with plane stepping occurring in the *j* direction
ISTEP	(FALSE)	=.true. for stepping in the *i* direction within the plane: e.g., it cannot=.true. if JKPLANE =.true.
JSTEP	(TRUE)	=.true. for stepping in the *j* direction within the plane
KSTEP	(FALSE)	=.true. for stepping in the *k* direction within the plane
INDQ	(1)	index of adaption flow-field variable, *q*; default of $1 \Rightarrow \rho$
IQ(8)	(0)	enables a combination of *q* variables to drive the adaption

RDSMAX	(2.0)	relative maximum allowed $\Delta s : \Delta s_{MAX} \geq 1.0$
RDSMIN	(.5)	relative minimum allowed $\Delta s : \Delta s_{MIN} \leq 1.0$
CLAM(1)	(.01)	$=\lambda$, coefficient of torsion parameter τ, $1 \leq \lambda \leq 10^{-6}$
CLAM(2)	(.0001)	$=\lambda^*$, coefficient of plane torsion parameter, ψ; same range as λ
CT(1)	(.5)	proportion of "straightness" to "orthogonal" for torsion vector $\bar{t}$
CT(2)	(.5)	as CT(1), but proportions plane torsion vector, $\bar{t}^*$
NEDGE	(0)	override control on side-edge adaption: 1=both edges, 2=start edge, 3=end edge
MG1	(0 or 4)	used with NEDGE: number of points to merge start-side spacing
MG2	(0 or 4)	used with NEDGE: number of points to merge end-side spacing
INTER	(2)	order of interpolation: 2, 3, or 4
NFILT	(2)	number of passes to filter (smooth) q and ω
MGSTEPS	(0)	number of merging lines for within-plane torsion controls
MGPLS	(0)	number of merging planes for between-plane torsion controls
MARCH	(FALSE)	=.true. to extrapolate end adaption *line* throughout remaining lines
MARCHPL	(FALSE)	=.true. to extrapolate last adapted *plane* throughout remaining planes
ADD	(0)	$=n$ to add n points between each node in selected range
LSTADD	(IST)	lower limit of range for adding points (i.e., when ADD>0)
LENDADD	(IEND)	upper limit (if ADD=0, values are ignored)
SUB	(0)	$=n$ to delete n points between each node in selected range
LSTSUB	(IST)	lower limit of range for point deletion (i.e., SUB>0)
LENDSUB	(IEND)	upper limit (if SUB=0, values are ignored)
REMOVE	(0)	removes requested number of points from outer grid region
NOUP	(FALSE)	=.true. if no adaption required (e.g., for moving outer boundary)
SAVE	(TRUE)	=.false. to suppress output of data files
ORTHS(2)	(TRUE)	=.false. to remove orthogonal constraint at start boundaries
ORTHE(2)	(TRUE)	=.false. to remove orthogonal constraint at end boundaries
GEOM	(FALSE)	=.true. to include geometry at wall boundaries in adaption variable
QFUN	(TRUE)	=.false. to remove $f(q)$ from adaption variable (used with GEOM=.true. only)
NOQ	(FALSE)	=.true. if the q file is not used
LNSING	(0)	$=n$ if the line n is common to all adaption planes
PLSING	(0)	$=n$ if adaption plane n is collapsed to a line
TWOD	(FALSE)	=.true. if datasets are 2-D (see section 2.4.1)
FV	(FALSE)	=.true. if q file in finite-volume format (2.4.2)
MGRID	(0)	adaption grid number if multiple grids
EXPORT	(FALSE)	denotes an export plane transfer, not an adaption pass, multigrid only
IMPORT	(FALSE)	denotes an import plane transfer, multigrid only
MPLANE	(1)	defines plane number, used for multiple grid export and import only
IS,IE	(0,0)	defines i direction domain for plane transfers
JS,JE	(0,0)	defines j direction domain for plane transfers
KS,KE	(0,0)	defines k direction domain for plane transfers
MVBOUND	(0)	nonzero to move the outer boundary
RECLUST	(0)	method to redistribute points, with or without boundary move
NSM	(10)	smoothing filter in the line-stepping direction for outer boundary
NSM1	(10)	smoothing filter in the plane-stepping direction for outer boundary
NOSESM	(0)	smoothing across planes with singular line at nose
DSW	(0)	wall spacing, only for RECLUST>0
DSE	(5.0)	outer edge spacing, only for RECLUST=1 or *imax*

DSW1	(0)	only for DSW=999, wall spacing at $j=1$
DSW2	(0)	only for DSW=999, wall spacing at $j=jmax$
DSN	(5)	percentage to expand when matching outer boundary to shock

First adaption Although many parameters have just been described, generally only a few are used for each adaption. The best technique, even for experienced users, is to retain as many of the default parameters as possible, view the results and then adjust some parameters if necessary. For some 3-D problems, it is clear which plane should be the stepping plane so the PLANE parameter, along with the STEP parameter, can be initially chosen. Note that only one PLANE parameter and one STEP parameter are needed for one adaption pass. Also, a flow-field variable should be chosen for INDQ that shows the flow features most clearly. To find appropriate within-plane parameters, adapt on only one plane and/or turn off the connectivity between planes (i.e., CLAM(2)=0) to see the set of 2-D adaption planes. If there are any lines common to all planes or if a plane is collapsed to a line, PLSING and/or LNSING must be set. If the first adaption pass is not acceptable, try increasing the ratio between RDSMAX and RDSMIN, decreasing CLAM(1), and/or setting NEDGE=1. Since many of the parameters have interdependent effects, it is better to change only one or two of the parameters at a time.

2.3.2 Explanation of user-supplied input parameters. The following is a detailed explanation of each of the input parameters; they are listed in alphabetical order.

ADD When this is nonzero, points are added between adjacent mesh points within the requested range (see LSTADD, LENDADD). For example, if ADD=2, two points will be added between each consecutive grid point. Adding occurs only in the adaption direction and not in either of the stepping directions. Ensure that the added points do not cause the coded array dimensions to be exceeded. ADD and SUB can be used in the same pass to move points: note that the ADD will occur before the SUB, so the range on the SUB parameter needs care.

CLAM CLAM(1)=λ and CLAM(2)=λ^* define the magnitude of the torsion parameters τ and ψ, respectively. As the values of these parameters decrease, more points will be pulled into the high-gradient regions, at the possible expense of grid smoothness.

CLAM(1) controls τ, the torsion parameter within the plane; its order of magnitude can lie between 10^{-6} and 1. A value of zero produces a set of independently adapted lines, possibly generating crossed grid lines. As λ increases, the grid becomes smoother but less adapted.

CLAM(2) controls the magnitude of ψ, the torsion between planes. It has the same range of values as CLAM(1); however, if it is zero, the adapted grid may still be acceptable. For periodic planes (i.e., if the first and last planes are the same), setting CLAM(2)=0 is necessary to prevent a discontinuous grid at the juncture.

CT CT(1)=C_t and CT(2)=C_t^*; they represent the direction of the torsion vectors $\bar{t}$ and $\bar{t}^*$ (whereas λ and λ^* are their magnitude in these directions). They have the range of $0 \le C_t, C_t^* \le 1.0$, where a value of zero emphasizes orthogonality and a value of one emphasizes straightness. The default of .5 places the torsion vectors halfway between. This value is suitable in most cases, but it may cause problems when side boundaries are concave or when adapting on already adapted planes.

DSE This is used only when RECLUST=*imax* (note: RECLUST=1 assumes RECLUST=*imax*) and provides the outer edge spacing. The Vinokur algorithm needs the first and last edge spacing along the recluster line. DSE is the multiple of the average mesh size Δs, and is defaulted to 5. Hence the last-edge mesh size input to the algorithm is DSE$*s_{max}/(npts-1)$ where s_{max} is the new s_{max} if the outer boundary location has changed. If 3>RECLUST<*imax*, the code computes the outer edge spacing to match across the boundary with the existing grid spacing and DSE is not used.

DSN When MVBOUND is based on a shock distance or a flow contour, the code finds the requested flow feature, s_{shock}, and places the new boundary point outside of it. DSN (default=5) determines the new boundary location as $s_{shock}+\text{DSN}*s_{max}/100.$, where s_{max} is the old s_{max}.

DSW This is the first mesh spacing at the inner boundary for the Vinokur algorithm (RECLUST≠ 0). If no value is input, the original Δs_1 is used. DSW can be very small if necessary, but if too small, there may be insufficient latitude to permit the algorithm to reach an appropriate solution downstream, creating a jagged grid. This problem may be resolved by providing a first and last wall spacing (DSW1 and DSW2). In this case, set DSW=999, and input DSW1 and DSW2. The code will compute DSW for each line.

DSW1, DSW2 These two parameters replace the single DSW and may resolve the jagged problem mentioned above. A very small value of DSW may be appropriate at the first active line *(j=1)* where s_{max} is small, but if the grid expands rapidly, the value of s_{max} downstream is too large to accommodate the same small wall spacing. In this case, DSW1 will be used at *j=1*, and DSW2 will be used at *j=jmax* and the intermediate values will be interpolated.

EXPORT For multiple grids only. When EXPORT is set to .true., the user-supplied parameters for this adaption pass describe the current location of a plane of data to be transferred from one location to another, most probably in another grid. This parameter set **must** be followed by an IMPORT parameter set that describes the plane's destination.

FORMI If equal to .true., the PLOT3D grid and Q input files (assigned to units 7 and 8) are assumed to be ASCII (formatted). Default is unformatted. Note that if SAGE is compiled as r8 (double precision), then **only** formatted files can be used.

FORMO If equal to .true., output files will be formatted (the unit numbers depend on the number of adaptions and multiple grid rereads). Default is unformatted.

FV This is set to .true. if the *q* file is in finite-volume format (i.e., the *q* variables are evaluated at the cell centers and not at the grid points). See section 2.4.2 for a more detailed description of the finite-volume option.

GEOM This parameter should be used when a wall boundary is defined by high surface gradients and the standard grid redistribution has moved points in such a way that the original shape has been deformed. When GEOM is set to .true., the code will add the surface curvature function to the flow-field gradient function in the wall boundary regions. Points will thus be maintained in or redistributed into regions of high surface curvature as well as into regions of high flow-field gradients. The contribution of the geometry function will proportionally decrease away from both boundaries, with the internal lines controlled by the flow field only. If GEOM=.true. and QFUN=.false., adaption will be to geometry gradients only, for all grid lines.

IJPLANE, IKPLANE, JKPLANE These parameters define the plane on which adaption takes place. By default, they also imply the direction of the stepping plane. The input parameter file needs only one of these parameters to be set to .true., and the code will automatically assign .false. to the other two. IJPLANE=.true. indicates that the plane represented by *(i,j)* will be the adaption plane and that plane stepping will occur in the *k* direction. Whether *k* is a forward or backward step will depend on KST and KEND (i.e., if KST > KEND then backward stepping will occur). Similarly, IKPLANE=.true. indicates that the adaption plane contains the *(i,k)* directions and that *j* is the plane-stepping direction. Finally, JKPLANE=.true. refers to the plane containing the points *(j,k)* and *i* is the plane-stepping direction.

IMPORT For multiple grids only. An input parameter list containing IMPORT=.true. must immediately follow a parameter set containing EXPORT=.true. The IMPORT list describes the destination (i.e., the receiving plane) of the plane of data defined in the EXPORT=.true. input-parameter set. The use of MGPLS is important for the subsequent adaption of the import grid.

INDQ This parameter indicates which of eight possible flow-field variable(s) will drive the redistribution of grid points. From the standard PLOT3D format, five options are available:

1 → density ρ
2 → x-momentum, ρu
3 → y-momentum, ρv
4 → z-momentum, ρw
5 → stagnation energy, e

Three more options are available by setting INDQ=6, 7, or 8

6 → pressure, p
7 → Mach number, M
8 → temperature ratio, T

Pressure, Mach number, and temperature ratio are computed using the ideal gas relationship and assumes the Q file contains the standard variables. (The code actually assigns pressure to NDIM+1, Mach number to NDIM+2, and temperature to NDIM+3, so if the user has changed the value of NDIM to accommodate extra flow-field variables, INDQ must reflect this change.) Note that INDQ=4 is not available for 2-D datasets. The user will normally choose to adapt to the flow-field variable that most represents the flow features. However, if different variables demonstrate different features, it may be advantageous to combine them to bring out all the features on the adapted grid. In this case, set INDQ=0 and input values for IQ.

INTER indicates whether to use a linear (INTER=2), a quadratic Lagrange polynomial (INTER=3), or a cubic spline (INTER=4) scheme for interpolations. Interpolation is used throughout the code; for example, the q values in the output function file are interpolated at the new adapted grid points. Linear interpolation will usually provide the appropriate result.

IQ is an array of eight (or NDIM+3) integer values that are used only when INDQ=0. They allow the user to modify the adaption variable to a combination of variables. The order of IQ is consistent with the order of the flow-field variables in Q. The value of an index is the proportion that the corresponding variable will contribute to the final adaption variable. For example, IQ(1)=1,IQ(7)=1 [i.e., (1,0,0,0,0,0,1,0)] will produce an adaptive function of $\frac{1}{2}(\frac{\partial \rho}{\partial s}+\frac{\partial M}{\partial s})$ and IQ(1)=1, IQ(3)=1, IQ(6)=3 will produce $\frac{1}{5}\frac{\partial \rho}{\partial s}+\frac{1}{5}\frac{\partial \rho v}{\partial s}+\frac{3}{5}\frac{\partial p}{\partial s}$. Obviously, IQ=(1,0,0,0,0,0,0,0) is the same as INDQ=1.

IST, IEND contain the indices defining the first and last boundary lines of the adaptive domain in the i direction. Similarly, **JST, JEND** define the domain in the j direction and **KST, KEND** define the domain in the k direction. These variables define the limits of the adaption domain and must lie within the input grid boundaries defined for the physical domain, i.e.,

$$1 \le \text{IST,IEND} \le \text{IMAX};\ 1 \le \text{JST,JEND} \le \text{JMAX};\ \text{and}\ 1 \le \text{KST,KEND} \le \text{KMAX}$$

Forward and backward stepping are also controlled by these parameters. If plane stepping is in the k direction, setting KST > KEND will produce backward stepping. Similarly, if stepping within the plane is in the j direction, setting JST > JEND will produce backward stepping. The reversing of the data is handled internally and is imperceptible to the user. Reversing either of the stepping directions will completely change the resulting grid, since the redistribution along the initial line and plane will be different, as will the connecting torsion springs. Reversing the order of the adaption direction (i.e., i in the default case) should have no effect on the solution, since the solution along a line is independent of the order of points. However, it can be used to redefine start and end points on a line if necessary, and for meshes that contain very large and very small Δs_i, numerical accuracy may be influenced by this adaption direction.

ISTEP, JSTEP, KSTEP These are used in conjunction with the PLANE parameters described above and define the marching and adaption directions within the defined plane. Only one of these three parameters should be input and set to .true., and the code will assign .false. to the other two. If IJPLANE=.true., then only ISTEP or JSTEP can be true (KSTEP must be false). If ISTEP=.true., stepping occurs in the i direction and thus the adaption direction will be j. Points will be adapted along each constant i line and stepping will occur to the next i line, forward or backward, depending on IST and IEND. If JKPLANE=.true., only JSTEP or KSTEP can be true and similarly, for IKPLANE=.true., only ISTEP or KSTEP=.true. will have any meaning. The code puts out an error message if these inputs are inconsistent.

IS, IE, JS, JE, KS, KE These parameters are used only for an export or import transfer process in a multiple-grid file. These variables define the range of the transfer domain within the plane, MPLANE. The PLANE parameter (e.g., IJPLANE) defines which coordinate plane is being transferred. As an example, if IKPLANE=.true., then IS, IE, KS, and KE are used to define the domain within the transfer plane. If they are omitted, it is assumed that the entire plane is being transferred.

LNSING In some 3-D grid types, planes emanate from a common grid line. If the chosen set of adaption planes includes this common line, then this line should be adapted only on the first plane and not

on subsequent planes. This adapted line is then placed in the first line of all planes. Although the input option is LNSING=n, where n is the line number, it is likely that $n=1$.

LSTADD, LENDADD These are input only if ADD≠0 and if only a portion of the grid is to be expanded. If they are not input and ADD≠0, the entire adaption domain (not physical domain) is assumed. If ADD=0, their values are ignored.

LSTSUB, LENDSUB These parameters are input only if SUB≠0; they define the limits in which points are to be removed. If they are not input, then the entire adaption domain is assumed.

MARCH This parameter refers to stepping within the plane. If the last line to be adapted (j_{end}) is within the physical grid boundary (i.e., $j_{end} < j_{max}$), a sharp discontinuity will occur between the last adapted line, j_{end}, and the nonadapted line, j_{end+1}. Setting MARCH=.true. causes the remaining lines within the plane (i.e., $j_{end+1} \rightarrow j_{max}$) to be realigned so that they are proportional to the last adapted line. This realignment will be performed for every plane.

MARCHPL This parameter refers to the plane-stepping direction, and can be used independently or in conjunction with MARCH. If the last adaption plane is within the physical boundary (i.e., $k_{end} < k_{max}$), each line in each subsequent plane will be proportioned with respect to the adapted lines in the k_{end} plane.

MG1, MG2 When NEDGE is nonzero, an override mesh spacing is computed at the requested boundaries (either first, last, or both). This edge-point spacing, which is not a function of the adaption but of the initial grid, is merged into the three adjacent points to produce a smooth transition. The default value of MG1 (for i_{st}) and MG2 (for i_{end}) is zero when NEDGE=0, but four when NEDGE is requested. The user has the option of overriding this value, setting it to any other integer value. This can be used when adapting a boundary layer; when either MG is increased, the dense edge spacing is maintained over a larger region. For example, MG1=10 would merge the edge spacing into the next nine cells.

MGPLS This input variable is analogous to MGSTEPS described below, but applies to planes. If the first adaption plane (k_{st}) is internal to the physical domain, MGPLS=n permits the user to gradually bring in the effect of the adaption parameters to produce a smooth transition across the start plane. No adaption will take place on the first plane (k_{st}), and after n planes, full adaption occurs with the adaption parameters C_t^*, λ^*, etc., equaling their input value. This is an especially important feature for grids that have an initial distribution on the wall boundary that defines a physical shape and cannot be changed. It is also useful for retaining matching multiple-grid boundaries (see IMPORT).

MGRID This parameter is only used for multiple-grid files and indicates which grid to adapt. Do <u>not</u> use MGRID=1 for a single grid: setting MGRID to nonzero automatically implies a multiple-grid file.

MGSTEPS (m_g) provides continuity when the first adaption line on each plane is internal to the physical boundary. Inputting MGSTEPS=n tells the code to start with no adaption on the initial line (i.e., retain the original distribution on the j_{st} line) and to linearly increase the adaption effect until, after n lines, full adaption occurs. At this point, C_t, λ, etc., will coincide with their input values. If MGSTEPS=1, no adaption will be performed on the first line, but full adaption will occur on the second line. Figure 9 shows an example with $m_g = 5$, and section 1.4.1 describes the parameter in detail.

MVBOUND This is the input parameter that invokes the outer boundary movement. There are three options:

(1) MVBOUND=±p, where p is a percentage of the input line length. For example, MVBOUND=−10. will create a grid where s_{max} along each line is 10% smaller than the initial grid.

(2) MVBOUND=999. will move the outer boundary based on the shock location s_{shock} of the specified variable, INDQ. The code will look for a sudden change in gradient of the flow variable and add a given distance to this shock location to create a new outer boundary.

(3) MVBOUND>1000. will allow the boundary to parallel a given contour value found in the Q file (defined by INDQ). The contour value is MVBOUND−1000.

MPLANE Used only for multiple grids during an import or export process. MPLANE is an integer defining the transfer plane. The PLANE parameter is used (e.g., IJPLANE=.t.) to indicate the coordinate direction of MPLANE.

NEDGE is a flag that requests an override on the computed side-edge boundary spacing. Side edges occur at i_{st} and i_{end} and frequently need special handling. If there are no flow gradients near the edge of the domain, the standard adaption algorithm will pull points away from the edge. This may not be a satisfactory result, as, for example, when the side edge is internal to the physical grid boundary. Figure 10(a) shows a side-edge adaption with NEDGE=0. The flow-field gradients are concentrated in the center of the grid, and the first adaption point *(i=4)* has been pulled far from the boundary line at $i_{st} = 3$. It is clear that it is preferable for the adapted side-edge spacing to be continuous with the juxtaposed spacing in the external region. Even if the two boundaries coincide (i.e., $i_{st} = 1$), the user may prefer a different spacing than that computed by the adaption algorithm. In either case, NEDGE can be set and the code will try to improve the side-edge spacing. The computed mesh-size override is merged into the next four points, but this number can be changed by MG1 and MG2. The result of setting NEDGE=1 is shown in Fig. 10(b). Depending on the case, both or only one of the side spacings may need improving. NEDGE=1 requests both edges, NEDGE=2 requests start edge only, and NEDGE=3 requests end edge only.

NFILT is a "filtering" variable that defines the number of passes used to smooth the gradient of the input q data and the computed tension parameter, ω. The default value of two will generally suffice, but if the flow-field variables are discontinuous, it may be helpful to increase the value of NFILT. An increase in NFILT can also be used to expand or spread out a very sharp flow feature.

NOUP This variable is used to change the grid (e.g., ADD, SUB, MVBOUND, RECLUST) without performing an adaption. NOUP stands for NOUPdate.

NOQ If no q file is available, SAGE can still be used to smooth the grid: perhaps to equal spacing or to the geometry function, or to recluster, with or without moving the outer boundary. If NOQ = .true., no file will be read on unit 8, and instead a constant flow field will be generated internally and there will be no interpolated solution file. This is quite different from QFUN = .false. where the q variables are not used, but are interpolated onto the new grid and output.

NOSESM This variable was added to remove a problem caused by a special grid structure that uses a singular line at the nose. Since the smoothing parameters (NSM, NSM1) fix the first and last points, s_{max} at the nose will remain fixed. NOSESM will change the nose s_{max} to give a smoother appearance. NOSESM is the number of adjacent points used to smooth s_{max} over the nose region. Typically the value should be 1, 2, or 3.

NSM, NSM1 NSM and NSM1 use the same filtering option described elsewhere (see NFILT and the *FILTER* subroutine). Since the outer edge of a shock may not be smooth (due to initial calculations and/or a coarse mesh) these two parameters will help smooth the new boundary, and though the default value is 10 in both cases, numbers up to 100 may be appropriate. NSM is used to smooth the *i=imax* for all *j* in the IJPLANE, whereas NSM1 smooths the cross plane (*i=imax* for all *k* in the IKPLANE). Occasionally this smoothing parameter overemphasizes an irregularity in the outer surface and should be set to zero if this occurs (see Fig. 35 in Section 3).

ORTHS, ORTHE The code assumes that orthogonality to the marching boundaries is desirable. This may not be the case, as, for example, in outgoing flow where the shape of the outer boundary is arbitrary. Setting ORTHS(1) = .false. will turn off orthogonality from the first to second adaption lines, and ORTHE(1) = .false. performs the same function when approaching the end adaption line. ORTHS(2) and ORTHE(2) will similarly affect the plane boundaries.

PLSING This is a parameter unique to 3-D grid adaption. It stands for plane singularity, and implies that a plane n is actually a single line, but is stored as a set of identical lines. The code will not be able to compute normals and will certainly "blow up" unless informed of this condition. It is only relevant when the adaption plane direction coincides with this collapsed plane. Do not use it when adapting in other plane directions.

QFUN This parameter permits the user to adapt to the geometry function only. When QFUN = .false. **and** GEOM = .true. are input, the coefficient of the flow-field gradient (C_q) is set to zero for all grid

lines (see section 1.5). In this case, the geometry function is computed throughout the grid and drives the adaption for all grid lines.

RDSMAX, RDSMIN control the density of the redistributed points and are the maximum and minimum allowable grid spacings. They are input as proportioned values and are changed to physical variables internally, i.e., $\Delta s_{MAX} \times s_{max}/(n_i - 1)$ and $\Delta s_{MIN} \times s_{max}/(n_i - 1)$, where n_i is a constant equal to the total number of points along the adapted line, and s_{max} is the length of the current adaption line. This implies that RDSMAX≥1.0 and RDSMIN≤1.0. (If both are set to 1.0, a uniform grid will result.) As an example, RDSMIN = 0.5 will prevent a converged Δs from being less than half the average step size. (Since many factors influence the distribution of grid points, this control is not absolute.) Note that since the adaption along the first line is not influenced by the torsion parameters, this initial line will present a clearer picture of the effect of RDSMAX and RDSMIN.

RECLUST This parameter is used to recluster the grid points, with or without boundary movement and is quite different from adaption. If outer boundary movement is requested (MVBOUND ≠ 0), the default of RECLUST = 0 specifies that the new spacing on the line is proportional to the original spacing. RECLUST = 1 will invoke the Vinokur algorithm and provide spacing based on an inverse hyperbolic function. If no outer boundary movement is desired (MVBOUND = 0), but reclustering based on the Vinokur algorithm is required (RECLUST = n, where $3 > n$), then n points will be reclustered. This is useful for limiting reclustering to the boundary layer or for entire grid reclustering. Note that this grid redistribution option is also useful for removing zero cells (i.e., two grid points at the same location) from an initial input grid.

REMOVE It is possible that an initial grid has unnecessary grid points in the outer region. Once an initial solution has been obtained, the user can see that these points are wasted. REMOVE=n will remove n points from the end of all adaption lines. NOUP should be set to true if only removing points is required. To remove points from the inner region, reverse IST and IEND. This will not change the adaption but will fool the remove operation. Remember, REMOVE deletes the boundary points, SUB retains the outmost one.

SAVE This is useful when more than one adaption pass is made in the same program run, for example, an adaption stepping in the j direction followed by one stepping in the i direction. The output grid and function files are large, and setting SAVE=.false. on a set of $NAMEL will suppress the output for that particular adaption. If SAVE=.true. (default), each subsequent output set of *xyz.out* and *q.out* files will be assigned to different unit numbers. As stated in the execution section, the first output set is assigned to units 10 and 11. The second output set will therefore be units 12 and 13, and so on. Note: SAVE=.false. cannot be used for multiple grids.

SUB When SUB=n (≠0), points are removed from the adaption line. As an example, if $n=1$, every other point is deleted. If $n=2$, two consecutive points are deleted between the points retained. Note that the number of stepping lines remains constant: points are only removed from the adaption line. To remove points from both directions, two passes are required. See LSTSUB, LENDSUB if only a selected range of deleted points is to be deleted.

TWOD If the input grid and function files are stored as 2-D PLOT3D files, this parameter must be set to .true. The code will assume that IJPLANE is the adaption plane and JSTEP the stepping direction. The user must specify ISTEP=.true. if required. The next section discusses the adaption of 2-D problems.

2.4 Alternative Grid Types

2.4.1 Two-dimensional adaption. The SAGE code can accommodate 2-D datasets, and will adapt the single plane in the same manner as in the original 2-D SAGE code (Davies and Venkatapathy, 1989). When the input parameter TWOD is set to .true., the code will read the grid (assuming *(x,y)*) and function files as 2-D files. These datasets will then be internally converted to a 3-D format; the number of k planes will be set equal to one, creating a constant z coordinate, and the 4th q function will be shifted to index 5. (Note that indices INDQ and IQ retain their 3-D relationship.) Because of this reorganization, no special handling of 2-D datasets is required within the body of the code. Datasets are reconverted to 2-D form before output. To accommodate larger dimensions, the parameter statement may be changed at the beginning of each routine. Since 2-D datasets require only KD=1, ID, JD, and IMX may be significantly

increased. However, this change is not made automatically and the user must change the parameter statements if necessary.

2.4.2 Finite-volume grids. The solution file associated with finite-volume applications contains q values evaluated at the cell center. Therefore the IMAX, JMAX, and KMAX values on the header record are one less than the values given in the grid file. If SAGE finds that the size records disagree and that the finite volume option is off, an error message is sent to the user.

Care must be taken with the boundary (or ghost) cells. SAGE interpolates q onto the internal grid points and then sets all boundary values (in the physical domain) equal to the adjacent interior value. After adaption, flow values are interpolated back to the cell center. If all planes are not adapted, the q values in the final adapted plane will be interpolated as if they are on a 2-D surface since there will be a discontinuity between the adapted and non-adapted grid points. The boundary values of the physical domain are again set equal to the adjacent values, regardless of whether the entire physical domain has been adapted. It is therefore very important for the user to check all the boundary cells in each coordinate direction.

Related to the handling of the boundary cells, SAGE will only adapt a finite-volume grid if it is a single surface (whether defined in 2-D or 3-D) or is a 3-D grid with four or more planes.

2.4.3 Multiple grids. The multiple-grid format is a single file containing a collection of separated grids. Preceding the grids are two header records, one defining the number of grids and the other the size of each grid. The associated q file is similarly defined.

Some complex multiple grids utilize the blanking feature available in PLOT3D. Currently, this version of the SAGE code does not handle these blanked regions (see Section 5). Also, if overlapping regions are adapted in separate grids, it is the user's responsibility to interpolate the results. However, matching zonal planes can be handled with the plane-transfer feature described below.

2.4.3.1 File handling. Since each grid is stored sequentially within a multiple-grid file, SAGE reads and copies all grids (and their associated solutions) to the appropriate output files until the requested grid (MGRID=n) has been read. This grid is now adapted (or a plane transferred) and written, along with the interpolated flow solution, into the correct sequence in the output files. Finally, any grids following the adapted grid are also read and copied to the output files. Subsequent adaption or export/import passes will use these output files as input files. If several passes occur in the same computer run, several sets of large files could be created and the user is reminded to delete unneeded files. The SAVE parameter cannot be used since the code needs to rewind the files for multiple passes.

An additional set of files will be created if any grid has changed size (e.g., the ADD or SUB option has been used). Grid sizes are stored in the header record that has already been written to the output file before adaption took place. This header record must be amended to reflect the new grid size. The final output files are therefore read in as input files, the header record is corrected, and yet another set of output files is created. An output message keeps the user informed of the unit numbers for the final set of output files.

2.4.3.2 Data transfer using EXPORT and IMPORT. Section 1.7 explains the need for data transfer between grids. A set of input parameters is available that controls the transfer of data from one domain within a specified plane of a specified grid to a matching domain within another grid. The same namelist format is used that controls the normal adaption procedure in SAGE, but two sets are required in the user-input file: the first describes the 'export' plane and the second describes the 'import' plane. Here, the term 'export' means a domain (i.e., a surface, plane, or sub-plane) whose data will be transferred. The domain receiving this data is the 'import' domain. A transfer domain is defined by

1. the transfer code, either *export=.t.* or *import=.t.*
2. the grid number, *mgrid*
3. the plane direction, (*ijplane,ikplane,jkplane*)
4. the plane number, *mplane* (this will often be the first or last plane)
5. the range of points describing the domain (i.e., the start and end points in two directions) within the plane

Notes: (a) For 2-D datasets only one line in a plane is transferred and *ijplane =.t.* and *mplane=1* are defaulted by the code. The transfer domain is described by *is, ie, js, je* and one of these directions must have equal start and end points. If the 'export' card is the first input set, remember to include *twod=.t.*

(b) Multiple transfers can be handled in the same run of the code as well as a combination of adaptions and transfers. See the examples section for clarification.

2.4.4 Blanked grids. SAGEv2B is the only version of SAGE that recognizes the blanking option that is found in some PLOT3D grid files as an additional variable, called IBL. Section 5 of this document describes the blanking option in detail. If IBL is defined in the PLOT3D input grid file, SAGEv3 can be used but the blanking will be ignored and the output grid file will not contain IBL.

2.5 Output Message File

The *sage.out* file is written to unit 6, the normal default for the output screen. It contains messages that help explain what has happened during program execution. At the end of each adaption, "**ADAPTION *n* COMPLETE**" indicates that the program was able to run to completion and that *xy.out* and *q.out* files have been created. The message "**OUTPUT FILES ON UNITS n_1 AND n_2**" informs the user that the output grid and Q files are named fort.n_1 and fort.n_2.

The following are other messages that may be seen (given in alphabetical order) along with a short description of their meaning.

ADD OPTION EXCEEDS DIMENSION (Critical)
Self-explanatory. Increase array dimensions.

CANNOT USE INDQ > NUMBER OF Q IN SOLUTION FILE (Critical)
This message comes from subroutine SHOCKLE. Only variables within the Q file can be used to detect a shock, not internally computed values.

FINITE VOLUME METHOD NEEDS 1 OR 4+ PLANES (Critical)
A single plane is treated like a 2-D surface: i.e., 4 points are used for cell-centering. Four planes are needed for the finite volume method in 3-D.

GRID HAS IDENTICAL POINTS AT *i* AND *i+1* ON LINE *j* AND PLANE *k*:
USE RECLUST OPTION TO TRY TO REMOVE (Critical)
Computation of body normals is impossible in cells of zero size. Use the reclust option to remove them, and perform the adaption again.

GRID SIZE TOO LARGE FOR THIS ADAPTION, MINIMUM DIMENSIONS REQUIRED: ID=n_1, JD=n_2, KD=n_3 (Critical)
Increase array dimensions to size suggested. Note that these values are appropriate for this set of input-control parameters only and may need to be changed for other adaptions of the same grid. For multiple grids, this message may be preceded by additional information.

IMPORT CARD EXPECTED, NOT FOUND (Critical)
The previous card in the input stream contained *export=.t.* This must be followed by a set containing *import=.t.*

INCONSISTENT PLANE AND STEP (Critical)
A stepping direction has been requested that is not available for the requested plane. For example, IKPLANE and JSTEP.

INPUT FILE SIZES DO NOT MATCH (Critical)
The grid dimensions on the header records of the grid file and solution file do not match.

IS THIS FINITE VOLUME? IF SO, SET FV=.TRUE. (Critical)
The grid file dimensions are one greater than the solution file. Should this be finite volume?

MAX I TOO LARGE, CHANGE ID TO n_1
MAX J TOO LARGE, CHANGE JD TO n_2
MAX K TOO LARGE, CHANGE KD TO n_3
CHANGE IMX TO n_4 (All critical)
These messages occur only for multiple grids. It implies that one or more of the grids in the file is too large for the dimension statement. It need not be the adaption grid. It is possible that even if this is

corrected, a subsequent run could indicate a second dimension change to accommodate data swapping for the adaption grid.

NDIM TOO SMALL FOR 2-D TO 3-D TRANSFORMATION (Critical)

Increase NDIM dimension (e.g., from 4 to 5) so internal transformation can be made.

NO CONVERGENCE ALONG INITIAL LINE, ERRMIN=a_1 (Warning)

The initial line is a 1-D adaption only. This is rarely a catastrophic error, especially if a_1 is small; however, the adaption may not be completely satisfactory. The only control parameters that affect the initial line are RDSMAX, RDSMIN, and NEDGE.

NO CONVERGENCE ON LINE *j* AND PLANE *k*, ERR=a_2 (Warning)

This message is only a warning and adaption continues. Even many of these messages may of be no concern as long as a_2 is small. If adaption is successfully completed, check the new mesh to see if it is acceptable.

NO OUTPUT FILES (Warning)

For whatever reason, no files have been output. (Is *save=.f.* on all passes? Does $namel start in column 2?)

NO POINTS ADDED (Warning)

Inconsistency in parameters requesting adding points.

NO VALUE FOUND FOR QVAL ALONG INITIAL LINE (Critical)

Outer boundary movement has been requested along a constant contour line. This value has not been found on the first grid line.

NUMBER OF POINTS INCREASED FROM n_1 TO n_2 (Informational)

If the ADD option has been input, this message gives the new grid dimensions.

NUMBER OF POINTS DECREASED FROM n_1 TO n_2 (Informational)

If the SUB option has been input, this message gives the new grid dimensions.

OUTPUT FILES ON UNITS n_1 AND n_2 (Informational)

The final grid file will be found on unit n_1 and the final Q file is written onto unit n_2. Note that intermediary files may also have been created. For the simplest single-grid, single-adaption case, default is fort.10 and fort.11.

PLANE n_1, GRID n_2 COPIED TO PLANE n_3, GRID n_4 (Informational)

Indicates the successful transfer of a surface from one grid to another in a multiple grid file.

POINT(S) REMOVED, NUMBER OF POINTS NOW *n* (Informational)

REMOVE option has been invoked.

s* IS NON MONOTONIC ON LINE *j* AND PLANE *k (Critical)

This message will terminate the program. It indicates that the values of s_i at the completion of the iteration on line *j* are not monotonically increasing, thus implying crossover of points. Since this is unacceptable, the program outputs the data. It is then possible to view the plots and re-evaluate the control parameters.

SOLUTION FILE IS F-V: CHECK YOUR BOUNDARY CELLS! (Informational)

Data stored into the ghost or boundary cells may be incorrect.

SUB OPTION PRODUCES TOO FEW POINTS FOR ADAPTION (Informational)

Fewer than 10 points remain in the adaption direction.

TOO FEW POINTS FOR ADAPTION WITH NEDGE=1 (Critical)

There are fewer than 10 points along the adaption line. This is not appropriate, especially if NEDGE is set.

TOO MANY MULTIGRIDS FOR NM DIMENSION (Critical)

The parameter statement at the beginning of each subroutine contains NM, the number of zones in a multigrid file. The default value is 10.

WARNING: DIRECTIONAL SIGNS INDICATE COORDINATE SYSTEM MAY NOT BE RIGHT-HANDED (Critical or informational)

The calculation of body normals assumes a right-handed coordinate system. In most cases, this message indicates the grid should be re-ordered.

WARNING: ZERO CELL AT (i,j,k), 'SAGE WILL TRY TO CORRECT!'

Routines SHOCKLE and BLCLUST detect zero cells during the reclustering process. This message is passed on to the user as a warning. Note that the *(i,j,k)* are in the transformed coordinate system. The code will average two surrounding cells.

2.6 Outline of Each Subroutine

The *MAIN* routine is a driver routine whose task is to call the relevant subroutines. A loop (using NADS) is set to provide for multiple adaption passes. The first routine to be called is *INITIAL*, which reads and organizes the data. A loop is then set up for the adaption of each plane, and the constant planar variables are computed. Finally, a loop is set up for each line in the plane in which all the coefficients of the adaption equation are computed, followed by the solution process. When each pass is complete, the *OUTPUT* routine is called.

Along with the *MAIN* routine, the SAGE code consists of the following subroutines, listed here (as they are in the source code listing) in alphabetical order. Arguments shown in boldface are computed within the subroutine.

ADDPTS(**IER**)

This routine is called by *INITIAL* if $\mathrm{ADD} \neq 0$. Extra points (depending on ADD) are inserted between every grid point within the range LSTADD to LENDADD, by linear interpolation. The error flag is set if the additional points exceed the defined dimensions.

ADDV(COSX1,COSY1,COSZ1,A1,COSX2,COSY2,COSZ2,A2,**COSX,COSY,COSZ**)

This is a utility routine. The direction cosines of two unit vectors (COSX1,COSY1,COSZ1) and (COSX2,COSY2,COSZ2) are input arguments. The routine computes the direction cosines (COSX,COSY,COSZ) of the unit vector that represents the sum of the input vectors, proportioned by the coefficients A1 and A2.

BLCLUST

This routine is called when the user requests a reclustering of points using the Vinokur algorithm but with no outer boundary movement (RECLUST=$n>3$). Frequently this is used to transform an inviscid grid into a viscous grid. New values of s_i are computed, and the *(x,y,z)* and Q values are interpolated. If only a subset of the domain is reclustered ($n<imax-1$) the mesh spacing at the boundary is defined by the original spacing and not the input value of DSE.

BLOCK(J,K)

Initially, all grid coordinates are stored in the input X, Y, and Z arrays. In this routine, a block of data around the current j line is stored in arrays XJ, YJ, ZJ to prevent the original grid from being overwritten during interim calculations. Only the converged adaption line is replaced into the original grid arrays. The XJ, YJ, and ZJ are made up of all i points on the current j line and all i points on the $j-1$ and $j+1$ lines in the $k-1$, k and $k+1$ planes, giving $(n_i,3,3)$ dimensioned arrays. At boundaries, nonexistent data points are filled with 999. Most calculations within the code, but especially the computation of the normal vectors, are performed on this block.

CLUST2(**X**,DX0,DX1,JMAX)

This computes the new s_i (X) based on the algorithm defined by Vinokur (1983). Also includes ASINHF and ASINF function routines.

CROSSV(XT,YT,ZT,XT1,YT1,ZT1,DST,COSV,**AAP,DAP,ICROSS**,J)

Appendix II, section 1.11.2, contains the analysis used to develop this routine. COSV is the array containing the direction cosines of the vector $\bar{v}$, defined in that appendix. (XT,YT,ZT) are the coordinates of a j line, (XT1,YT1,ZT1) are the coordinates of a juxtaposed line, and DST is Δs along j. As an example, this routine is used to compute $s-s'$ (i.e., AA') and DA', the distance between s' and the

corresponding node at *(i,j–1,k)*, as shown in Fig. 4. ICROSS is the array containing *1* and indicates the intersecting segment for each COSV.

CSPLIN(NT,S,V,**SPF**)

The cubic spline coefficients, SPF, are computed for NT points. S is the streamwise location of the function V. These coefficients are used by the routine *SPEVAL* if GEOM=.true. or INTER=4.

DETERM(A1,B1,C1,A2,B2,C2,A3,B3,C3,**DET**)

This routine computes the determinant of the three vectors whose direction cosines are given in the argument list.

DLENG(JL,K)

When NEDGE is set, the tension parameter ω is amended at the edges (i.e., at IST and IEND) to improve edge-boundary spacing. This routine computes DLENGS and DLENGE, which are used in the edge ω calculation (by the routine *WTEDGE*). The values of DLENGS and DLENGE depend on whether the grid is defined outside of the adaption domain. JL indicates which *j* line is needed.

*EDGEMG(*VAR)

This is a utility routine. For various reasons, the values of some variables at the IST and/or IEND edges are overridden. To blend these different values into the calculations, this routine will merge the new values (given at the two boundaries of VAR) into the next three (or MG1, MG2) grid locations of VAR.

FBAR(J,K)

This routine is called once for every *j* line. The Δs and the gradients $\partial q/\partial s$ are computed at the input grid points and stored in FQ. If duplicate points are found (i.e., $\exists$ a $\Delta s_i = 0$) a message is printed and the program terminates. If GEOM=.true., the wall gradients $\partial g/\partial s$ are also computed by calling *WALLS*, and stored in FG. In addition, the coefficient C_g is computed and FG is added to FQ and stored in F. The routine *INTF* is called to interpolate the value of F at these new grid points and to compute the normalized form of F, i.e., FB. The routine *GETB* is called to find the value of B for this *j* line. For lines other than the first, initial guesses for the *s* distribution are extrapolated from the converged s_i along the *j–1* line. Since these do not correspond to the input points, new local values of *x, y,* and *z* are interpolated by calling *PROPS*.

FILTER(VAR,NIPTS,NFILT)

This routine smoothes the parameter contained in VAR by adding a second derivative term, $v_i = .75v_i + .125(v_{i+1} - v_{i-1})$. NFILT is the number of smoothing passes (default=2). The parameters smoothed are $f_i = f(\partial q/\partial s)$ and ω_i; they are returned to the calling routine via VAR.

FVORG(IND)

If IND=1, this routine interpolates the cell-centered *q* values onto the grid points to enable adaption to take place in the normal manner. Values at the first and second boundary points are set equal. If IND=2, the *q* values are re-interpolated to the new cell centers.

GETB(J,K)

This routine computes the value of B used to evaluate ω. B is found by an iterative process and is said to converge when the minimum requested Δs equals the computed minimum Δs. The analysis for this routine is given in Appendix I.

GETWT

This routine computes ω_f, the modifier of ω that is applied when any computed Δs lies outside the requested range of RDSMAX and RDSMIN.

*HEADIO(*IC,**IER**)

This routine reads (IC=1) and writes (IC=2) the header records for multiple-grid files. It also tests to see if the grid dimensions are too large for the programmed dimensions and whether the NM dimension parameter is greater than or equal to the number of zones.

INITIAL(**NOMORE,MTCH**)

This routine sets all the default values and reads the input parameter file. If EXPORT is true for a multiple-grid file, MTCH is set to .true. and control is handed to the *MATCH* routine. In all other cases, the

appropriate grid and function file input routines are called (either *READAT* or *READMULT*). When necessary, the grid points and corresponding flow-field data are rearranged to correspond to the data organization assumed by the analysis. This routine also controls the addition or removal of points and initiates any boundary movement. If no more adaptions are requested, NOMORE is set and control is returned to the main routine.

INTF(F1,**F2**,S1,SMID,NPTS)
This routine interpolates to find F2 (new F) at the new s_i (S1), based on the input values of F1 (current F) and the midpoints (SMID) of the input s array. The interpolation routine chosen is based on INTER.

INTXYZQ(J,K,J1,K1,SS,SN, **QJ**)
This routine interpolates for X, Y, Z, and Q at the new SN, given the corresponding values at SS. The new *(x,y,z)* coordinates are stored in the block of data defined by XJ, YJ, and ZJ. The q data are stored in QJ. The appropriate interpolation routine is called based on INTER.

LAGCOF(SNEW,SARR,NPTS,**M,P1,P2,P3**)
This routine computes the Lagrange coefficients P1, P2, and P3 for a point SNEW, with respect to the input s array, SARR. These are used by the calling routine to interpolate for the variable at SNEW. First or second order is available; the choice depends on INTER. M is the associated index for P1 and will reflect a forward or backward interpolation, depending on the location of SNEW within the interval.

LINE1
This routine solves for the adapted values of s_i along the initial line $j = j_{st}$ on the initial plane k_{st}. Since both torsion terms are zero on this line, the Δs_i are computed from the 1-D approach.

MARCHJ(K)
If the final adapted line within plane k is internal to the physical end-boundary line, this routine will redistribute the points on the remaining j lines, based on the distribution along the j_{end} line. The routine is called for each plane. This action is performed only on request by the user and is not an adaption to the flow field. However, it will prevent the discontinuity between the last adapted line and the remaining non-adapted grid lines.

MARCHK
The function of this routine is similar to *MARCHJ* but it is called on user request if the final adaption plane k_{end} is internal to the physical end plane k_{max}. It will redistribute points on all lines on the remaining planes to be proportional to the corresponding line on the last adapted plane.

MATCH(**NOMORE,IER**)
This routine is called as soon as an 'export' card is read. If necessary, the output files are rewound (*REWND*); and then the header records are read (*HEADIO*). The export grid is read into core (*MULTIO*) and the requested plane stored (*STOREX*). Now, the next set of input parameters is read (which should be an 'import' set). The input files are closed, rewound, and reread up to the import grid (*READMULT*). The stored export plane is copied into the import plane (*STORIM*) and all the files are written to the output units (*WRITMULT*).

MGWALLS(J)
This routine is called when GEOM=.true. It computes the coefficient of the geometry function, C_g (FGW), based on the local aspect ratio.

MULTIO(IN1,IN2,IOUT1,IOUT2)
This routine reads and immediately writes multiple grid files. IN1 and IN2 give the range of grids to read in (e.g., if MGRID=3, *READMULT* will initially call *MULTIO* with IN1=1 and IN2=3). IOUT1 and IOUT2 give the range of grids to output (in the same example, *READMULT* will set IOUT1=1 and IOUT2=2, since grid 3 will be written after adaption). The header records are handled separately in *HEADIO*.

NOADAPT(J,K)
This routine updates variables and/or places them in appropriate arrays when no adaption is to be performed on the current j line, but adaption is to be performed on the next line. When stepping to the

next line, the code expects certain variables to be available at *j–1*. This scenario occurs, for example, in merging situations (MGSTEPS>0) and for common lines (LNSING>0).

NORM(F1,NPTS)

The function F1 is normalized as $(F1 - F1_{min})/(F1_{max} - F1_{min})$. If maximum and minimum values are equal or nearly equal, the normalized variable is set to $O(10^{-5})$.

NORMPT(IP,JP,KP,INDPL,**PA,PB,PC,SING**)

This routine finds the vector at the point (IP,JP,KP) normal to the plane defined by INDPL. Although three direction planes exist through the point, only two are needed by the calling routines: INDPL=1 is the *(i,k)* plane and INDPL=2 is the *(i,j)* plane. The analysis to describe this routine is given in Appendix III. Four normals are computed and the average is found, with the direction cosines returned in (PA,PB,PC). SING is set to 1 if the normal does not exist.

OUTPUT

OUTPUT is called at the conclusion of each adaption set (NADS). Remaining planes are proportioned if requested by *MARCHK*, and then the data files are returned to their original order to conform with the input mesh structure. Either *WRITMULT* or *WRITOUT* is called to output the grid and flow-field files. This routine is also called if *s* becomes non-monotonic (OK=.false.). In this case, the new mesh points that have been computed are output to help the user choose more appropriate control parameters.

PROPS(J,K)

After a new solution of s_i has been obtained on a line *j*, the code stores this data in line *j–1* and steps to the next line. This routine proportions the new s_i throughout the non-adapted regions of the block of data defined by XJ,YJ,ZJ. Essentially, this provides a first guess for the current *j* line and also produces smoother planes for the vector normal calculations (see Appendix III).

PURPLE(A1,A2,A3,B1,B2,B3,**V1,V2,V3**,NFL)

A and B are direction cosines of two vectors defining an enclosed plane. This routine takes their cross-product and normalizes the result to give the direction cosines (V1,V2,V3) of the unit normal to the enclosed plane. NFL is the direction sign of the normal.

PUSHIT(DSN,NSM,NSM1)

This routine controls the outer boundary movement for all *j* at each *k* plane. It is called if MVBOUND $\neq 0$. It first calls SHOCKLE to find the new s_{max}. If $d_w = 999.$ the original wall spacings are saved for each *j*. The location of *(x,y,z)* at the new s_{max} are computed, and then the new array of s_i is obtained, either as proportional to the old s_i or by calling *CLUST2*. Finally, all *(x,y,z)* and *q* are interpolated at the new s_i.

READAT

This routine reads in the single grid and function files in PLOT3D binary format. Input datasets may be in 2-D or 3-D form. The size of IMAX, JMAX, and KMAX on the header record is checked (by *SIZING*) before the data is read. If NOQ is true, the Q array is filled with 1.0.

READMULT(**IER**)

This is the read routine for multiple-grid files. Since some writing of files also occurs, the output unit numbers are verified. The header records are read and copied to the output files (*HEADIO*). Then *MULTIO* is called to read the requested grid into core and to write all preceding grid data onto the output files. Finally, if NOQ is true, the Q array is filled with 1.0.

REWND

This routine rewinds the output files and opens them as input files.

SETUPJ(J,K)

This routine computes the direction cosines of the vectors $\bar{u}$, $\bar{b}$, and $\bar{e}$ used to evaluate the torsion vector $\bar{t}$. These vectors are associated with the *(i,j)* points within the constant *k* plane.

SETUPK(J,K)

This routine is similar to *SETUPJ*, but the direction cosines of $\bar{u}^*$, $\bar{b}^*$, and $\bar{e}^*$ are computed to evaluate the plane torsion vector, $\bar{t}^*$. These vectors are associated with the *(i,k)* points within the constant *j* plane.

SHOCKLE(**SNMAX**,DSN,NSM,NSM1)
This routine is called when the outer boundary is moved, and it computes all the new s_{max} for each *(j,k)*, based on shock location or other user request. Smoothing also takes place, based on input NSM and NSM1.

SINGPLN
When PLSING is set, *SINGPLN* stores the result of the first adapted line into every line of the same plane.

SIZING(**IER**)
The parameter statement at the beginning of each subroutine presets the dimensions (ID,JD,KD) of the grid and *q* files. This routine compares the input grid dimensions (IMAX,JMAX,KMAX) or {IM(NGRID),JM(NGRID,KM(NGRID)} to these preset values. If insufficient space has been allocated, the minimum possible values of ID, JD, and KD are computed for this adaption to proceed. (These values are not obvious since space must be allocated for data swapping.) IMX, the maximum of ID, JD, and KD is also evaluated. A message is then sent to the user to recompile SAGE with the suggested dimensions. Finally, IER is set to 1 to inform the *INITIAL* routine to terminate the code.

SOLUT(J,K)
By the time *SOLUT* is called, all variables have been computed that are needed to obtain the new distribution of s_i. The coefficients of s_i (see Eq. (8) in the first section of this report) are set up in a tridiagonal matrix and solved for s_i. Interpolated values of ω_i are found at these new values of s_i and iterations are performed until $\sum\left|s_i^{(n)} - s_i^{(n-1)}\right|$ is small or too many iterations have been performed. In both cases, a check is made to see if all the s_i are monotonic. If so, the program continues; if not, the flag OK is set to false, causing the program to output the current grid and terminate.

SPEVAL(NT,S,V,SPF,SI,**VI,VPI,VPPI**)
This is a cubic spline interpolation routine used when INTER=4 or if GEOM is true. It uses the coefficients (SPF) computed in *CSPLIN*. In addition to interpolating for the variable V at SI, the corresponding first (VPI) and second (VPPI) derivatives are also evaluated.

STOREX
This routine stores the data from the export plane into XP, YP, ZP, and QP.

STORIM
The export data stored by *STOREX* in XP, YP, ZP, and QP are copied to the import plane.

SUBPTS
When $\mathrm{SUB} = n \neq 0$, n points are deleted between each retained mesh point. If LSTSUB and/or LENDSUB are nonzero, the range of the deletion is restricted. Deletion occurs only in the adaption direction and does not decrease the number of stepping lines or planes.

SWAPINV
This routine is called if $k_{st} > k_{end}$, $j_{st} > j_{end}$, or $i_{st} > i_{end}$. The order of *(i,j,k)* in the *x*, *y*, and *z* input matrices is reorganized to ensure that internal computations have monotonically increasing indices. The flag for the handedness of the coordinate system is amended accordingly.

SWAPXYZ(RSWAP)
Since the internal computation assumes that *j* is the stepping direction within the plane and that *k* is the stepping direction of planes, this routine is called to interchange *x*, *y*, *z*, and *q* when the input requests alternative stepping directions. RSWAP is a flag that states whether this data exchange is at the start of the computations or is the reverse process required for output. This routine is lengthy due to minimizing storage requirements.

SWAP2D(IO)
TWOD=.true. indicates that the input datasets are formatted in two dimensions. This routine reorganizes the 2-D plane to appear in the code as a 3-D surface. Every *z* is given the value of zero, each Q(4) is moved to Q(5), and Q(4) is zeroed. IO indicates whether the translation is from 2-D to 3-D, which occurs on input, or from 3-D back to 2-D for output.

TORCOF(L,JK,JKST,JKEND,MGNOS,MARCHJK)

TORCOF is called twice in each loop, once with all the arguments representing the k plane passing through *(i,j,k)* and once with the arguments representing the j plane also passing through *(i,j,k)*. This routine amends the coefficients (C_t, λ) etc. of the torsion vectors $\hat{t}$ and $\hat{t}^*$ based on the current line location. For example, C_t is decreased when leaving or approaching a boundary to emphasize orthogonality.

TORSION(J,K)

This routine first chooses the appropriate $\hat{b}$ and adds to $\hat{u}$ to obtain $\hat{n}$. The torsion vector $\hat{t}$ is then obtained by adding $\hat{n}$ and $\hat{e}$. The between-plane torsion vector $\hat{t}^*$ is computed in a similar manner. The routine *CROSSV* is then called to find the intersection of the torsion vectors with the j line from which both s_i' and s_i^* can now be evaluated. Finally, a check is made to ensure that s_i' and s_i^* monotonically increase. If they do not, the code attempts a correction, but any major problems will cause the code to terminate in the *SOLUT* routine.

UNITV(X1,Y1,Z1,X2,Y2,Z2,**DIRCX,DIRCY,DIRCZ**)

This is a utility routine that finds the unit vector from (X1,Y1,Z1) to (X2,Y2,Z2). DIRCX, DIRCY, and DIRCZ are the direction cosines of this vector.

UPDATE(J,K)

This is the last routine called in the iteration loop for the current j. Newly adapted values of s_i have been found. The values of x, y, z, and q at this new distribution are interpolated (*INTXYZQ*) and replaced into the matrices containing the physical mesh.

VMERGE(DIRV,LST,LEND)

This routine performs the same function as *EDGEMG*, but with a vector quantity (DIRV) in place of a scalar value. LST and LEND indicate which value of DIRV must be merged into the next three (or MG1, MG2) points.

WALLS(JW,K)

Along line j (JW), the geometry gradient, as defined by the radius of curvature, is computed for each grid segment. Normally, j will equal j_{st} or j_{end}. However, for cases when geometry is the only adaption variable (QFUN=.false.), *WALLS* is called for every j line.

WRITMULT

This is the write routine for multiple-grid files. The current adapted grid is now output and any subsequent grids are read and written by calling *MULTIO*. If the grid size has changed, the header record is updated; the output files are closed and opened as input files; the header record is corrected and all data are copied to the new output files.

WRITOUT

This routine writes the single-grid 2-D or 3-D adapted grid and interpolated function file on units NITG and NITQ. If the finite-volume option is set, IMAX, JMAX, and KMAX are one less on the q file.

WTEDGE(J,K)

This routine is called by *MAIN* when NEDGE modification is requested. The edge values of the tension parameter at the next j line are a function of the average $\omega\Delta s$ along the just-completed adapted line. This routine calls *DLENG* to obtain the appropriate value of edge Δs on the next line, computes the average $\omega\Delta s$ on this line, and evaluates WDS and WDE to be used in routines *LINE1* and *SOLUT*.

2.7 Nomenclature

The following is a list of variables used in the SAGE analysis. When applicable, the corresponding FORTRAN name used in the code is shown in boldface.

A, B	constants used to compute ω, **(A,B)**
A_s	aspect ratio, used to control C_g, **FGASP**
C_g	coefficient of $f(g)$ in ω calculation, **FGW**

C_q	coefficient of $f(q)$ in ω calculation, **FQW**
C_t	input proportion coefficient for torsion, **CT(1)**
C_t^*	input proportion for torsion between planes, **CT(2)**
$\vec{b}; (b_{x_i}, b_{y_i}, b_{z_i})$	orthogonal vector to j line within k plane; direction cosines of $\vec{b}$, **COSB**
$\vec{b}^*$	orthogonal vector to j line within j plane, **COSBK**
$C_{t_m}, C_{t_m}^*$	modified values of C_t, C_t^*, **CTM(1), CTM(2)**
$\vec{d}; (d_{x_i}, d_{y_i}, d_{z_i})$	straightness vector, **COSD**
$\vec{e}; (e_{x_i}, e_{y_i}, e_{z_i})$	average straightness vector within k plane, **COSE**
$\vec{e}^*$	average straightness vector within j plane, from $k-2 \rightarrow k-1$, **COSEK**
f	gradient of q (and g if necessary), **F**
f_{min}, f_{max}	minimum and maximum f used to normalize f, **FMIN, FMAX**
$\bar{f}$	normalized function of f, **FB**
g	geometry function
i	subscript indicating the current node in adaption direction, **I**
$imax, jmax, kmax$	total number of points in i, j and k directions of input grid file, **IMAX, JMAX, KMAX**
i_{st}, j_{st}, k_{st}	start of adaption domain in i, j and k directions, **IST, JST, KST**
$i_{end}, j_{end}, k_{end}$	indices indicating end of adaption domain, **IEND, JEND, KEND**
j	subscript of the current stepping line, **J**
k	subscript of the current adaption plane, **K**
κ	torsion-related constant
l	local subscript relating to node i, **L**
m_g	number of stepping lines before full adaption, **MGSTEPS**
m_g^*	the plane equivalent of m_g, **MGPLS**
n_g	flag for edge control, **NEDGE**
n_i	number of points in the adaption line, **NIPTS**
n_m	merging coefficient for lines $f(m_g)$, **CNM(1)**
n_m^*	merging coefficient for planes $f(m_g^*)$, **CNM(2)**
$\vec{n}; (n_{x_i}, n_{y_i}, n_{z_i})$	orthogonality vector within plane, **COSN**
$\vec{n}^*$	orthogonality vector between planes, also stored in **COSN**
q	input flow-field variable $(\rho, \rho u, \rho v, \rho w, e)$, **Q**
R	radius of curvature for geometry function, **FGS**
s	streamwise length, **SS** or **SN**
$\vec{s}; (s_{x_i}, s_{y_i}, s_{z_i})$	vector representing s, **SSX, SSY, SSZ**
s_{max}	maximum value of s on line j, **SMAX**
s'	value of streamwise length used for torsion within planes, **SP**
s^*	value of streamwise length used for torsion between planes, **SPP**
Δs_i	$s_i - s_{i-1}$, **DS**
$\Delta s_{MIN}, \Delta s_{MAX}$	requested minimum and maximum grid spacing, **DSMIN, DSMAX**
$\Delta s_{min}, \Delta s_{max}$	computed minimum and maximum grid spacing
T	torsion force
$\vec{t}; (t_{x_i}, t_{y_i}, t_{z_i})$	within-plane torsion vector, **COST**
$\vec{t}^*$	torsion vector between planes, also **COST**
t_n	proportion of $\vec{b}$ and $\vec{u}$ used to compute $\vec{n}$, **TN(1)**
t_n^*	proportion of $\vec{b}^*$ and $\vec{u}^*$ used to compute $\vec{n}^*$, **TN(2)**
$\vec{u}; (u_{x_i}, u_{y_i}, u_{z_i})$	vector normal to $j-1$ line in k plane, **COSU**

$\vec{u}^*$	vector normal to j line on $k-1$ plane, **COSUK**
x,y,z	input grid mesh, **(X,Y,Z)** globally and **(XJ,YJ,ZJ)** locally
λ	input magnitude of torsion control parameter, **CLAM(1)**
λ^*	input magnitude of plane torsion control parameter, **CLAM(2)**
θ	angle for torsion computation
τ	within-plane torsion-related parameter, **TAU**
ω	tension spring force, **WEIGHT**
ω_t	computed weighting on ω, **WT**
ψ	between-plane torsion-related parameter, **TAUPL**

2.8 List of Major Variables

This section contains the list of variables (in alphabetical order) used in the SAGE code. Local variables that contain only intermediary values are not listed. The format is:

Variable name(dimension) $/r_1/r_2/$brief description

where r_1 describes what type of variable — input, local, parameter, or common block name
and r_2 lists routine(s) where the variable is initialized.

A	/COM2/INITIAL/ A used to compute ω
AA(IMX)	/local/SOLUT/ coefficient of s_{i-1} in solution matrix
AAP(IMX)	/argument/CROSSV/ $s-s_i'$ or $s-s_i^*$
ACT	/local/TORSION/ final modified C_t
ADD	/COM19/input/ if set, add grid points
ALPHA	/COM12/input/ information only for PLOT3D
AMACH(IMX)	/local/FBAR/ computed Mach number
B	/COM2/GETB/ B used to compute ω
BB(IMX)	/local/SOLUT/ coefficient of s_i in solution matrix
BCONV	/local/GETB/ convergence criteria for B iteration
BJ1	/local/GETB/ value of B along $j-1$ line in the k plane
BK1	/COM15/GETB/ value of B along j line in the $k-1$ plane
CC(IMX)	/local/SOLUT/ coefficient of s_{i+1} in solution matrix
CLAM(2)	/COM10/input/ λ and λ^*, magnitude of torsion terms
CLAMW(2)	/COM10/TORCOF/ modified CLAM
CNM(2)	/COM10/TORCOF/ λ modifiers for MGSTEPS $\neq 0$ and MGPLS $\neq 0$
CONV	/COM15/INITIAL/ general convergence criteria
COSB(IMX,3)	/COM7/TORSION/ direction cosines of $\vec{b}$, in *(i,j)* plane
COSBK(IMX,3)	/COM20/TORSION/ direction cosines of $\vec{b}^*$, in *(i,k)* plane
COSD(IMX,3)	/local/SETUPJ,SETUPK/ temporary straightness vector, $\vec{d}$
COSE(IMX,3)	/COM7/SETUPJ/ straightness vector, $\vec{e}$, in *(i,j)* plane
COSEK(IMX,3)	/COM20/SETUPK/ straightness vector, $\vec{e}^*$, in *(i,k)* plane
COST(IMX,3)	/local/TORSION/ torsion vector, $\vec{t}$ or $\vec{t}^*$
COSU(IMX,3)	/COM7/SETUPJ/ $\vec{u}$, normal to $j-1$ line, in k plane
COSUK(IMX,3)	/COM20/SETUPK/ $\vec{u}^*$, normal to j line in $k-1$ plane
CT(2)	/COM10/input/ C_t and C_t^*, directions for torsion vectors
CTM(2)	/COM10/TORCOF/ modified C_t and C_t^*
DAP(IMX)	/COM9/CROSSV/ DA' for s' calculation
DAPPL(IMX)	/COM9/CROSSV/ DA' for s^* calculation
DET	/argument/DETERM/ value of three-order determinant
DLENGE	/COM6/DLENG/ Δs computed for end-edge modification

DLENGS	/COM6/DLENG/ Δs computed for start-edge modification
DMINSDB	/local/GETB/ $\partial min(\Delta s)/\partial B$
DSN	/argument/input,PUSHIT/ parameter for outer boundary location
DS(IMX)	/COM3/FBAR,LINE1,SOLUT/ Δs
DSE	/COM18/input/ outer edge spacing (when RECLUST $\neq 0$)
DSMAX	/COM6/FBAR/ Δs_{max} from RDSMAX
DSMIN	/COM6/FBAR/ Δs_{min} from RDSMIN
DSW	/COM18/input/ wall spacing for Vinokur algorithm
DSW1,DSW2	/COM18/input/ first and last wall spacing when DSW=999
EXPORT	/COM21/input/ if true, store data for transferring between multigrids
F(IMX)	/COM2/FBAR/ flow gradient, $f = \partial q/\partial s$, at input s
FB(IMX)	/COM2/INTF/ $\bar{f}$, normalized f at current s
FF(IMX)	/local/SOLUT/ coefficient of RHS of solution matrix
FG(IMX)	/COM17/FBAR/ normalized $\partial g/\partial s$
FGASP	/local/MGWALLS/ function of aspect ratio
FGS(IMX)	/COM17/WALLS/ $\partial g/\partial s$ at SMSS
FGW	/COM17/MGWALLS/ coefficient of FG for ω calculation
FORMI	/COM24/input/ true if ASCII (formatted) input files
FORMO	/COM24/input/ true if ASCII (formatted) output files
FQ(IMX)	/COM17/FBAR/ normalized $\partial q/\partial s$
FQW	/COM17/MGWALLS/ coefficient of FQ for ω calculation
FSMACH	/COM12/input/ information only, for PLOT3D
FV	/COM15/input/ flag to indicate finite-volume q file
GEOM	/COM11/input/ request for geometry function
ICROSS	/argument/CROSSV/ index for interval location of s' or s^*
ID	/dimension/parameter statement/ grid dimension in i direction
IEND	/COM4/input/ last node along i in adaption domain
IINVERSE	/COM13/INITIAL/ adaption requested in backward i steps
IJPLANE	/COM14/input/ true if *(i,j)* is adaption plane
IKPLANE	/COM14/input/ true if *(i,k)* is adaption plane
IM(NM)	/COM21/input/ IMAX for each grid in multigrid file
IMAX	/COM4/input/ number of points in physical domain, i direction
IMAXQ	/COM24/input/ IMAX for q file
IMPORT	/COM21/input/ if true, describes location to place EXPORT data
IMQ(NM)	/COM24/input/ IMAX for each grid in the multigrid q file
IMX	/dimension/parameter statement/ maximum value of ID,JD,KD
INDQ	/COM5/input/ index for q for adaption variable
INTER	/COM11/input/ order of interpolation, 2, 3, or 4
IQ(NDIM+3)	/COM5/input/ related to INDQ, combines q adaption function
IS,IE	/COM23/input/ i range of *export/import* transfer plane
IST	/COM4/input/ first node along i in adaption domain
ISTEP	/COM13/input/ true for stepping in i direction within the plane
JD	/dimension/parameter statement/ grid dimension in j direction
JEND	/COM4/input/ last node in j adaption domain
JINVERSE	/COM13/INITIAL/ adaption requested in backward j steps
JKPLANE	/COM14/input/ true if *(j,k)* is adaption plane
JM(NM)	/COM21/input/ JMAX for each grid in multigrid file
JMAX	/COM4/input/ number of points in physical domain, j direction
JMAXQ	/COM24/input/ JMAX for q file
JMQ(NM)	/COM24/input/ JMAX for each grid in the multigrid q file
JS,JE	/COM23/input/ j range of *export/import* transfer plane

JST	/COM4/input/ first node in j adaption domain
JSTEP	/COM13/input/ true for stepping in j direction within the plane
KD	/dimension/parameter statement/ grid dimension in k direction
KEND	/COM4/input/ last node in k adaption domain
KINVERSE	/COM13/INITIAL/ adaption requested in backward k steps
KM(NM)	/COM21/input/ KMAX for each grid in multigrid file
KMAX	/COM4/input/ number of points in physical domain, k direction
KMAXQ	/COM24/input/ KMAX for q file
KMQ(NM)	/COM24/input/ KMAX for each grid in the multigrid q file
KS,KE	/COM23/input/ k range of *export/import* transfer plane
KST	/COM4/input/ first node in k adaption domain
KSTEP	/COM13/ true for stepping in k direction within the plane
LENDADD	/COM19/input/ start of add points range
LENDSUB	/COM19/input/ start of delete points range
LNSING	/COM15/input/ flag to indicate a common line for each plane
LSTADD	/COM19/input/ end of add points range
LSTSUB	/COM19/input/ end of delete points range
MARCH	/COM14/input/ true to interpolate up to physical boundary
MARCHPL	/COM14/input/ true to interpolate to final plane
MAXITS	/COM15/INITIAL/ maximum number of iterations for convergence
MG1	/COM9/INITIAL/ number of merging points for NEDGE at i_{st}
MG2	/COM9/INITIAL/ number of merging points for NEDGE at i_{end}
MGCT(NM)	/COM16/INITIAL/ counter on number of adaptions per grid
MGPLS	/COM11/input/ number of planes before full adaption within plane
MGRID	/COM12/input/ multiple grid number for adaption
MGSTEPS	/COM11/input/ number of lines before full adaption within plane
MPLANE	/COM23/input/ plane number for export or import plane
MSI,MSJ,MSK	/COM21/SIZING/ contains overall maximum size of multiple grids
MTCH	/argument/INITIAL/ flag to indicate export and import has occurred
MVBOUND	/COM18/input/ invokes outer boundary movement
MXFG	/COM17/MGWALLS/ location of maximum $\partial g/\partial s$
NA	/dimension/parameter statement/ maximum number of adaption passes
NADS	/COM4/MAIN/ index on number of adaptions
NDIM	/dimension/parameter statement/ no. of q variables, normally set to 5
NEDGE	/COM9/input/ initiates side-edge boundary override
NFILT	/COM11/input/ number of passes for smoothing data
NFLAG	/COM15/INITIAL/ indicates direction sign for normal vectors
NGRID	/COM21/input/ number of grids in multigrid file
NIPTS	/COM4/INITIAL/ total number of i points in computation domain
NITGI	/COM12/INITIAL,WRITOUT.../ unit number for input grid file
NITGO	/COM12/INITIAL,WRITOUT.../ unit number for output grid file
NITQI	/COM12/INITIAL,WRITOUT.../ unit number for input q file
NITQO	/COM12/INITIAL,WRITOUT.../ unit number for output q file
NM	/dimension/parameter statement/ maximum number of multigrids
NOMORE	/argument/INITIAL/ no more input datasets remain; end program
NOQ	/COM11/input/ false to indicate no q file will be input
NOSESM	/COM15/input/ points to smooth nose kink after moving boundary
NOUP	/COM11/input/ prevents adaption, i.e., no update
NSM,NSM1	/argument/input,PUSHIT/ outer boundary smoothing
OK	/COM14/SOLUT/ flag to indicate normal termination
ORTHE(2)	/COM16/input/ false removes orthogonality at outer boundaries
ORTHS(2)	/COM16/input/ false removes orthogonality at initial boundaries

P1,P2,P3	/argument/LAGCOF/ coefficients of Lagrange polynomials
PLSING	/COM15/input/ flag to define a singular plane
PRES(IMX)	/local/FBAR/ computed pressure
Q(ID,JD,KD,NDIM)	/COM1/input,UPDATE/ flow-field variables
QFUN	/COM11/input/ false to allow geometry function only
QP(IMX,IMX,NDIM)	/COM23/STOREX/ q data of stored plane for grid transfer
RDSMAX	/COM5/input/ relative value of Δs_{max}
RDSMIN	/COM5/input/ relative value of Δs_{min}
RE	/COM12/input/ not used, PLOT3D variable
RECLUST	/COM18/input/ invokes Vinokur reclustering algorithm
REMOVE	/COM18/input/ number of points to remove at outer boundary
RSWAP	/argument/INITIAL,OUTPUT/ flag for swapping data
SAVE	/COM14/input/ flag to suppress output
SING	/argument/NORMPT/ flag to indicate nonexistent normal
SMS(IMX)	/COM3/FBAR/ midpoints of Δs
SMSS(IMX)	/COM17/WALLS/ s_i for geometry calculation
SN(IMX)	/COM3/FBAR/ current s array along j line
SNM(IMX)	/COM3/UPDATE/ converged s array along $j-1$ line
SNMK(IMX)	/COM3/FBAR/ converged s array along j line on $k-1$ plane
SP(IMX)	/COM9/TORSION/ s', intersection of torsion vector and j line
SPF(IMX)	/argument/CSPLIN/ cubic spline coefficients
SPPL(IMX)	/COM9/TORSION/ s^*, intersection of plane torsion vector and j line
SS(IMX)	/COM3/FBAR/ initial value of s along j line
SUB	/COM19/input/ request to delete points
TAU	/local/SOLUT/ τ, torsion parameter within plane
TAUPL	/local/SOLUT/ ψ, torsion parameter between planes
TIME	/COM12/input/ not used, PLOT3D variable
TN(2)	/COM10/INITIAL/ proportion of $\vec{b}$ to $\vec{u}$ for torsion vectors
TNM(2)	/COM10/TORCOF/ TN modified for boundaries
TRAT(IMX)	/local/FBAR/ computed temperature ratio
TWOD	/COM13/input/ indicates 2-D input files
WDE	/COM2/WTEDGE/ weighting factor used when NEDGE set
WDES	/COM6/WTEDGE/ retained WDE
WDS	/COM2/WTEDGE/ as WDE, but at initial boundary
WDSS	/COM6/WTEDGE/ retained WDS
WEIGHT(IMX)	/COM2/LINE1,SOLUT/ ω, tension parameter
WT(IMX)	/COM6/GETWT/ ω_t, correction to ω
WTSUM	/local/GETB,LINE1,WTEDGE/ $\sum 1/\omega$
X(ID,JD,KD)	/COM1/input/ input grid points, i direction
XJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ X data stored in block
XP(IMX,IMX)	/COM22/STOREX/ stored X of export transfer data
Y(ID,JD,KD)	/COM1/input/ input grid points, j direction
YJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ Y data stored in block
YP(IMX,IMX)	/COM22/STOREX/ stored Y of export transfer data
Z(ID,JD,KD)	/COM1/input/ input grid points, k direction
ZJ(IMX,3,3)	/COM8/BLOCK,INTXYZQ/ Z data stored in block
ZP(IMX,IMX)	/COM22/STOREX/ stored Z of export transfer data

3. EXAMPLES

This section contains many examples to familiarize the user with the adaptive-grid process. Each example includes plots of the initial grid and flow-field contours, the input-control-parameter file used to adapt the grid, the resultant adapted grid, and a discussion on the choice of control parameters. In addition, some cases show the improved flow solution obtained from the flow solver using the adapted grid as input. The 2-D examples show the effect of within-plane parameters that are also fundamental to 3-D problems, so 3-D users should also study these examples.

3.1 Two-Dimensional Examples

The first set of examples is for 2-D problems. Although the code adapts the data as if it were a single plane in a 3-D environment, this is imperceptible to the user. When TWOD=.true. in the input parameter file, SAGE will expect the data to be in the 2-D format of PLOT3D. The index of (1) on the CLAM and CT parameters refers to within-plane variables. Index (2) is not needed for these 2-D cases. If TWOD=.true. on the first pass, it will be assumed for subsequent passes in the same run.

Case 1. Flow in a Supersonic Inlet

Figure 13(a) shows the 101×79 initial grid (assigned to unit 7) for an inlet flow-field problem. Flow is from left to right (defined as the i direction) and a shock emanates from the upper-wall corner, reflecting off the lower wall. In addition, an expansion fan originates from the downstream upper-wall corner and interacts with the reflected shock. An interim computed solution (input on unit 8), generated by the flow solver using this initial grid, is shown as density contours in Fig. 13(b).

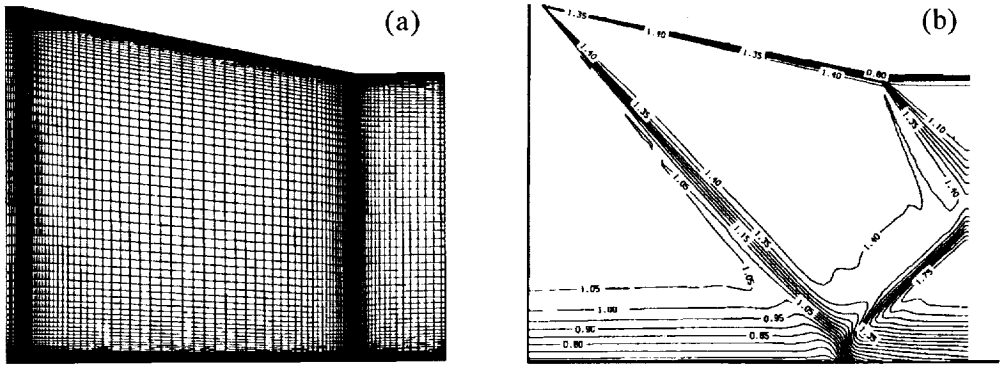

Figure 13. Flow in a supersonic inlet. (a) Initial grid; (b) computed density contours.

The SAGE code is now run to create a new grid that is more adapted to this solution, i.e., the grid points are redistributed with respect to a chosen flow-field variable and output to the file assigned to unit 10. SAGE also interpolates the complete flow-field solution onto these new grid points and outputs this file to unit 11. The updated files will subsequently be input into the flow solver to produce more accurate solutions. Several examples of grid adaption are given for this inlet problem to demonstrate the effect of varying the control parameters. Each example uses the grid and solution files shown in Fig. 13 as input.

Example 1: Single pass, stepping in *j* direction. Until the user is familiar with the basic parameters, the first attempt should be an input-control file with no parameters; i.e., all default values are used (with the exception of TWOD=.true.):

$namel twod=.t. $

On completion of the execution, the message to unit 6 states "ADAPTION 1 COMPLETE". The adapted grid using the default input parameters is shown in Fig. 14(a). The grid has been more evenly spaced and there is a slight clustering of points around the shock on the lower-wall boundary but the remaining grid lines are not adequately adapted to the flow features. This implies that the torsion term (whose magnitude is controlled by CLAM(1)) is too large and is overriding the local tension term, preventing the tension forces from pulling the points to the high-gradient regions. In addition, the minimum grid spacing is too large. Based on this information, a new input-control file was chosen:

$namel twod=.t.,rdsmin=.25,clam(1)=.001,nedge=1 $

Only three control parameters are input: the minimum allowable grid spacing, the magnitude of the torsion term (an order of magnitude less than the default value), and a request for edge control (which is frequently used). The remaining parameters retain their default values, signifying that the full grid will be adapted, density is the adaption variable, and stepping is in the *j* direction.

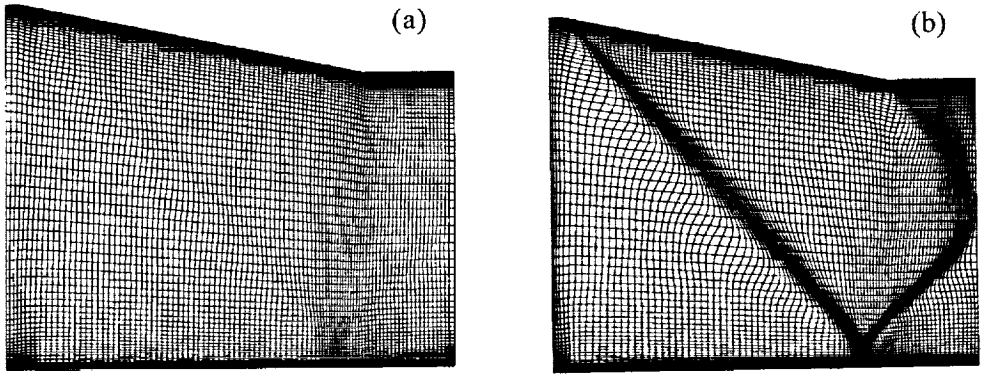

Figure 14. Adapted grids. (a) All default parameters; (b) adaption from bottom to top: rdsmin=.25, clam(1)=.001, nedge=1.

The result of this adaption is shown in Fig. 14(b). It can be seen that, since adaption began along the lower surface (*jst=1* is the default), the redistributed points cluster around the point of reflection on this line. However, points do not become sufficiently clustered at the corner shock and at the start of the expansion wave region on the upper surface. This is to be expected, since the adaption on the initial line is a 1-D solution and will pick up features quite clearly. However, as stepping continues, the features on subsequent lines get "dampened" by the torsion (i.e., smoothness) control.

Example 2: Adaption with backward *jsteps*. This example maintains the same parameters as Ex. 1, except that the upper surface is chosen as the initial adaption line, i.e.,

$namel twod=.t.,rdsmin=.25,clam(1)=.001,nedge=1,jst=79,jend=1 $

Figure 14(c) shows the result of this adaption. In this case, the upper surface is more clearly adapted, and the point of reflection on the lower surface is more spread out. Comparing these two examples indicates quite clearly that the starting line has a strong effect on the resultant adapted grid, and for some applications it should be chosen carefully.

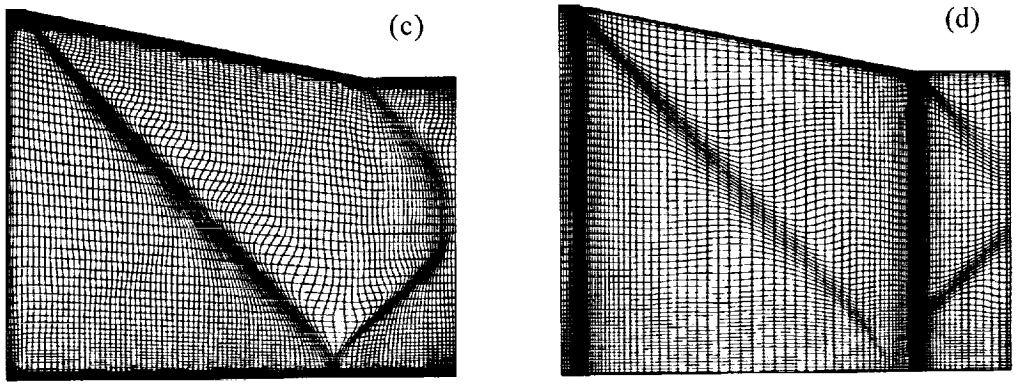

Figure 14 concluded. (c) Marching from top to bottom; (d) marching in i, from right to left.

Example 3: Adaption in *i* direction. This example shows the very different grid generated when the adaption is performed by stepping in the *i* direction (*istep=.true.*). Based on the experience obtained from the first two examples, adapting from right to left will produce a better grid since there are no gradients on line *ist=1* to adapt to. Hence, the input control file was chosen to contain:

$namel twod=.t.,istep=.f.,ist=101,iend=1,rdsmin=.25,clam(1)=.001,nedge=1 $

Figure 14(d) shows the result of this adaption. Not surprisingly, the reflected shock region on the lower surface is not "captured" by this adaption, and thus this grid is less suitable than those shown above. This demonstrates that the choice of stepping direction is important in producing a desirable grid.

Example 4: Effect of changing control parameters. This example is actually a collection of examples that show the effect of varying the control parameters *clam(1), ct(1), rdsmax,* and *rdsmin*. Figure 15(a) shows an adapted grid using "baseline" values. These are the same as the default values in the code with the exceptions of *clam(1)=.001* and adaption proceeds from top to bottom (i.e., *jst=79, jend=1*).

$namel twod=.true.,clam(1)=.001,jst=79,jend=1,nedge=1 $

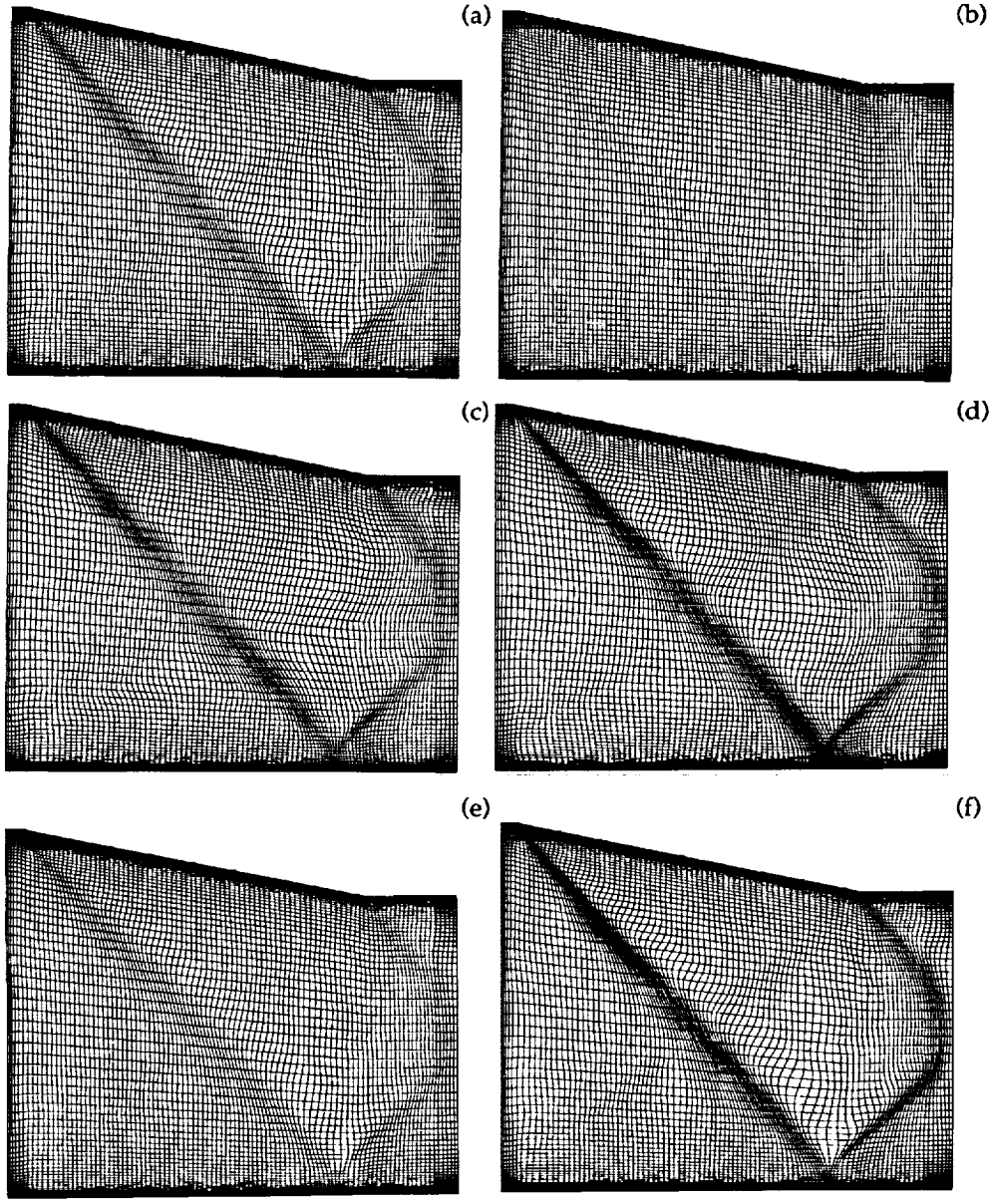

Figure 15. Effect of control parameters. (a) Baseline: clam(1)=.001,ct(1)=.5,rdsmax=2.0,rdsmin=.5; (b) clam(1)=.01; (c) clam(1)=.0001; (d) ct(1)=1.0; (e) ct(1)=.0; (f) rdsmax=4.0,rdsmin=.25.

Figures 15(b) through 15(f) are grids adapted by changing just one of the baseline parameters. Figure 15(b) shows the result with *clam(1)=.01*, and it is immediately obvious why this code default value was not chosen as the baseline value for these comparison cases: *clam(1)* is too large to allow any of the flow features to be "captured" by the adaption. Figure 15(c) shows the case for *clam(1)=.0001*, and this smaller value produces a grid with more points clustered around the shocks. Figures 15(d) and 15(e) show the effect

of the *ct(1)* parameter: *ct(1)=1.0* in Fig. 15(d), emphasizing straightness, and *ct=.0* in Fig. 15(e), emphasizing orthogonality. The latter shows how the orthogonality term dampens the adaption for this case: we are requesting orthogonality to the already parallel *i* lines. Finally, the effect of changing the minimum and maximum allowable grid spacing is shown in Fig. 15(f), where *rdsmax=4.0* and *rdsmin=.25* are used. This mesh size control is only a factor of two different from that in the baseline example (Fig. 15(a)), but significantly densifies the spacing in the shock regions.

Example 5: Two-directional adaption. Two-directional adaption is created by adapting in one direction (e.g., stepping in *j*) and then adapting in the other direction (i.e., *i*), using the first adapted grid as input. The resultant grid will depend on the order of the stepping direction. Two passes (i.e., two sets of input control parameters) were made to produce the adapted grid shown in Fig. 16(a). If both these passes are made during the same execution run (not to be recommended for a first attempt), then additional output files will be created on units 12 and 13. Data will be output for both passes, unless the SAVE parameter is set to false on the first set of control parameters. The control file for this example contains:

$namel twod=.t.,clam(1)=.001,rdsmax=4.0,rdsmin=.25,jst=79,jend=1,nedge=1 $
$namel istep=.t.,ist=101,iend=1,clam(1)=.001,rdsmin=.25,nedge=1 $

The first pass steps in the *j* direction and uses the same input control parameters as the adapted grid shown in Fig. 15(f). The second pass steps in *i*, and uses the same parameters as Ex. 3. Note that any parameters that remain unchanged for the two passes still need to be redefined in the control file, since the code restores all default parameters at the conclusion of each pass (with the exception of *twod=.true.*). The control file that produces Fig. 16(b) contains the same parameters, but the adaption order is reversed. That is, the first pass steps in the *i* direction and the second pass in the *j* direction. The difference between Figs. 16(a) and 16(b) is obvious and shows that the initial grid completely effects the adaption. Although this example shows that both adaptive sequences do adapt the grid quite reasonably, it should be noted that it is also possible that one order of adaption will produce a completely unacceptable result, while the reverse is quite suitable.

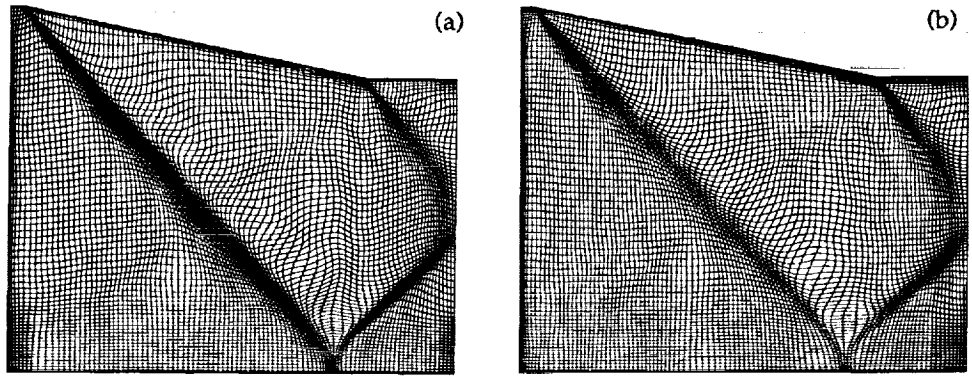

Figure 16. Two-directional adaption. (a) Marching in j followed by marching in i; (b) marching in i followed by marching in j.

Example 6: Flow solution using adapted grid. Figure 17 demonstrates the improved flow-field solution obtained when the flow solver is rerun using an adapted grid as input. The adapted grid shown in Fig. 17(a) was obtained by choosing the 'best' parameters, based on the experience already described:

$namel twod=.true.,jst=79,jend=1,rdsmax=4.0,rdsmin=.25,clam(1)=.0001,nedge=1 $

This grid and the interpolated flow-field variables were input to the flow solver, producing the flow solution (shown as density contours) seen in Fig. 17(b). The improvement in the resolution of the incident and reflected shock is obvious when compared to the initial solution shown in Fig. 13(b).

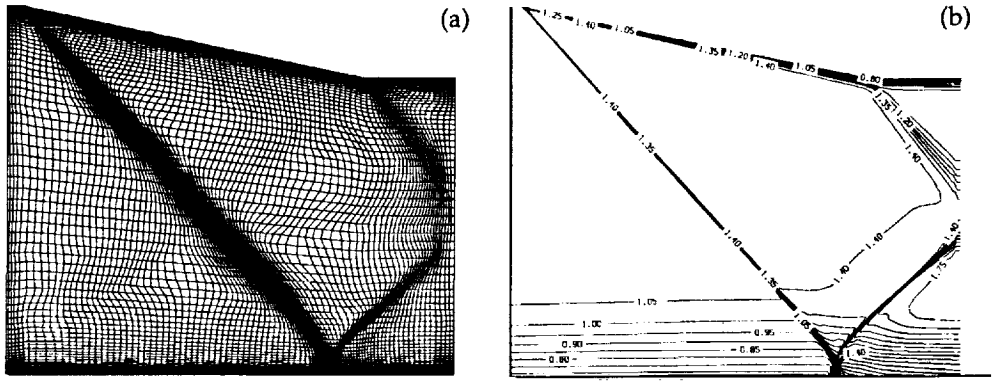

Figure 17. Solution using adapted grid as input. (a) 'Best' adaption, rdsmax=4.0,rdsmin=.25, clam(1)=.0001; (b) solution density contours, using 'best' adapted grid.

Case 2. Hypersonic Blunt-Body Flow

The second case contains two examples that demonstrate the effect of the CLAM and CT parameters. Figures 18(a) and 18(b) show an initial grid (32×32) and corresponding density flow-field contours for a hypersonic blunt body. The *j* direction marches out from the surface to the free-stream boundary, and the *i* direction marches along the body starting at the lowest point. This is a simple one-directional problem with the shock shape aligned with the grid. The outer-side boundary is curved (when marching in the *i* direction) and the default values of *clam=.01* and *ct=.5* will prevent the adapted grid lines from "turning" sufficiently at this edge, giving either a false densing of points at the outer side boundary or, possibly, a critical error message. In this situation, two remedies are available. CLAM can be decreased, hence de-emphasizing the effect of torsion and thereby allowing the tension force to "pull" the nodes toward the shock wave. Alternatively, the orthogonality restraint can be removed by setting *ct=1.0*. Figure 18(c) shows the result of setting *clam=.001* and retaining all other parameters at their default value.

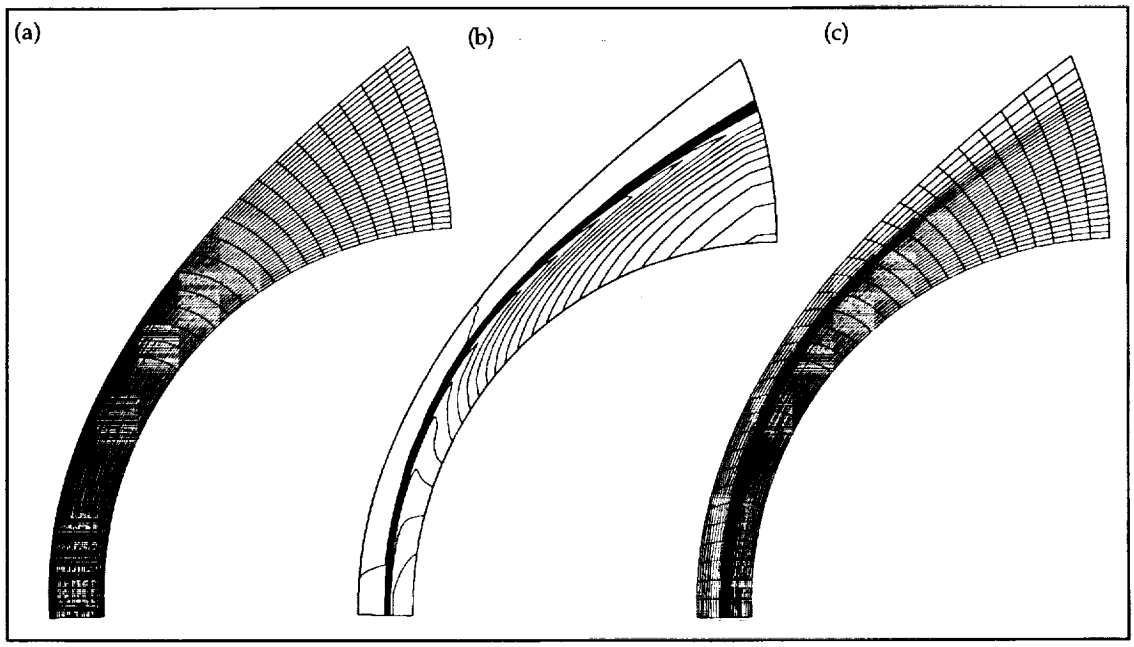

Figure 18. Hypersonic blunt body. (a) Initial grid; (b) initial density contours; (c) adapted grid, clam=.001.

The adapted grid obtained by setting *ct=1.0* is very similar to that shown in Fig. 15(c). The following are the two input-control files:

$namel twod=.t.,istep=.t.,clam(1)=.001$ and *$namel twod=.t.,istep=.t.,ct(1)=1.0 $*

Both adapted grids show the required result, i.e., the clustering of grid points across the shock.

Case 3. Blunt-Body Shock Impingement Problem

The third case introduces the use of the IQ and ORTHE parameters. Figure 19(a) shows the input grid of a cowl/lip shock interaction problem. The *j* direction marches from the body surface to the outer free stream and the *i* direction marches from the lower point of the sphere. The flow-field contours of both density (Fig. 19(b)) and Mach number (Fig. 19(c)) are given, since the adaption flow-field parameter is a combination of the two. The blunt-body shocks, impinging shock, and shear layers represent a complex flow that requires adaption in both directions to adequately capture all the flow features. Figure 19(d) shows the adapted grid (output on unit 13) produced by the following two-pass control file:

$namel twod=.t.,istep=.t.,ist=91,iend=1,indq=0,iq(1)=2,iq(7)=1,rdsmin=.25,
ct=.7,clam(1)=.005,nedge=2 $
$namel indq=0,iq(1)=2,iq(7)=1,rdsmin=.3,orthe=.f. $

The first pass steps in the *i* direction, starting from the topmost boundary. The parameter *indq=0* indicates that IQ will be input, and in this case two-thirds of the density function and one-third of the Mach function will be combined to become the adaption variable. As explained in case 2, the curvature of the outer boundary requires increasing CT. Setting *nedge=2* requests that only the side-edge boundary at *j=1* be treated. The second pass introduces the use of the ORTHE variable: stepping occurs from the sphere wall to the outer curved boundary, where the default would turn the grid to be normal to this outer boundary. ORTHE overrides this and allows the adaption to occur naturally.

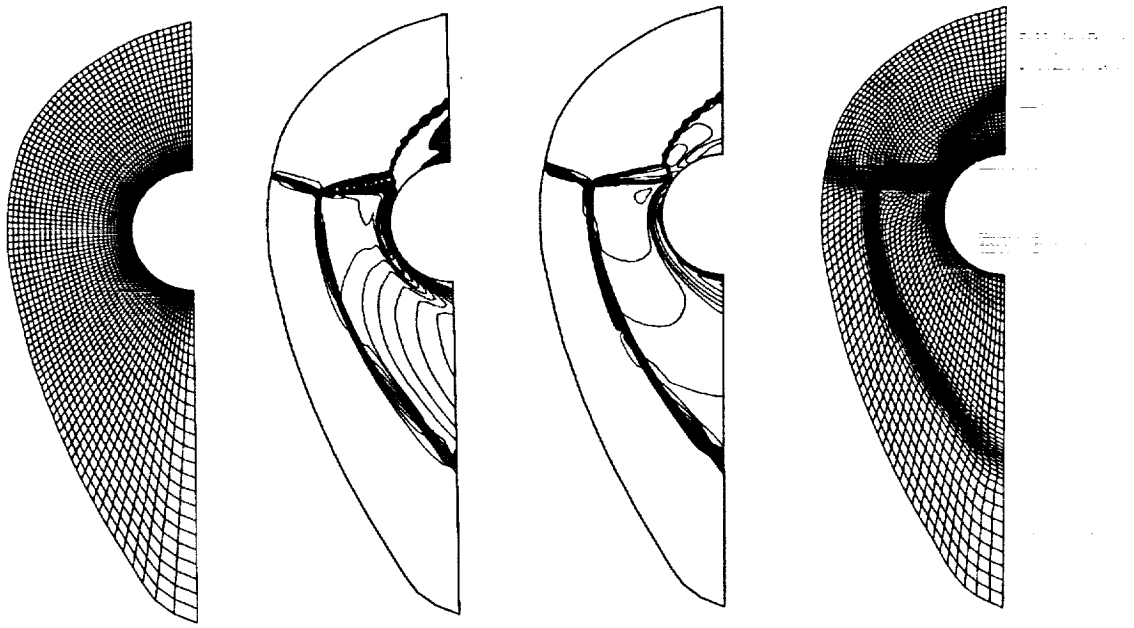

Figure 19. Blunt-body shock impingement problem. (a) Initial grid; (b) initial density contours; (c) initial Mach contours; (d) adapted grid.

Case 4. Hypersonic Inlet (Zonal Adaption)

Figures 20(a) and 20(b) show the initial grid and density contours for a hypersonic cowl/lip inlet problem. This is a more complex case and requires dividing the adaption domain into two zones: the blunt body region and the rectangular inlet region. The upper wall is the *j=1* line and the outgoing channel region (on the right side of the diagram) is the *i=1* line.

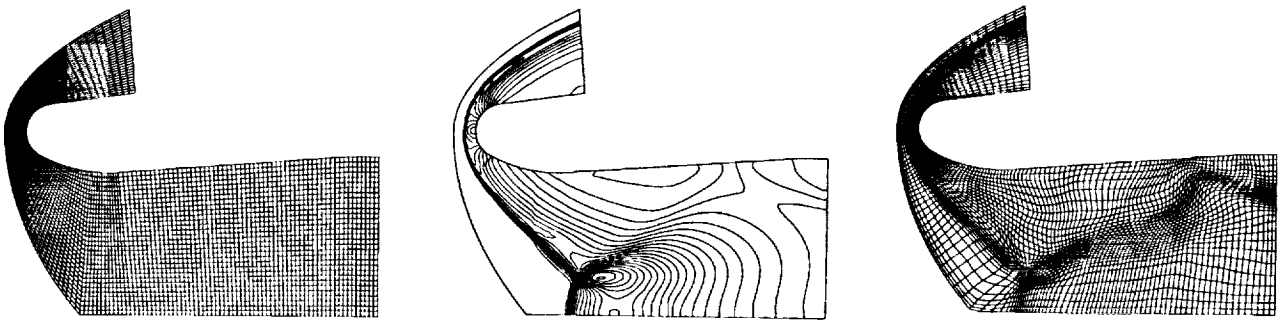

Figure 20. Hypersonic inlet, zonal adaption. (a) Initial grid; (b) initial density contours; (c)adapted grid.

Five adaption passes are required to create the final adapted grid shown in Fig. 20(c). The input control file consists of

```
$namel  twod=.t.,rdsmin=1.0,rdsmax=1.0,save=.f. $
$namel  jstep=.f.,rdsmin=.2,nedge=1,save=.f. $
$namel  ist=70,save=.f. $
$namel  jst=32,jend=1,iend=71,rdsmin=.25,clam(1)=.02,nedge=1,save=.f. $
$namel  iend=85,jst=19,jend=1,clam(1)=.002,nedge=1,mgsteps=5 $
```

The first pass, with equal maximum and minimum spacing, spreads out the i grid lines evenly; the original grid points were more densely distributed in the curved section, leaving fewer grid lines in the inlet area. This is a good example of using the program to improve an initial grid, with no flow-field adaption involved. The second pass steps in the i direction and adapts easily throughout the entire grid. The third, fourth, and fifth passes perform the adaption in the j direction. The very different flow features in the blunt-body region (blunt-body shock) compared to the features in the inlet region (Mach stem and reflected shocks) indicate dividing the adaption domain into these two zones: $i \leq 70$ and $i>70$. Only the blunt-body section ($i>70$) is adapted in pass three, with all default-control parameters. The adaption in the inlet domain starts at $jst=32$ in order to pick up the Mach stem along the lower wall. After stepping through the triple-point region (the intersection of the cowl shock and the reflected normal shock), the redistributed points do not spread out sufficiently, so the adaption is stopped at line 19 and a fifth pass is started at line 19 with a decreased value of CLAM. Since this pass starts internally, it is necessary to set MGSTEPS in order to merge smoothly from the already adapted region. Note: *save=.f.* has been used on the first four passes to prevent excessive file generation.

Case 5. Subsonic Impinging Jet

This example is to show the use of the SUB parameter. The original grid, with 231×100 grid points, and the corresponding Mach contours are shown in Figs. 21(a) and (b). It was demonstrated that adapting the grid enabled the user to decrease the density of grid points, and thus speed up the flow solver. Adapting the grid and reducing the number of points can be accomplished in the same execution run. The first step is to ensure that the dimension parameters are increased to ID=231, JD=231, KD=1, and IMX=231. (Note that SAGE will inform the user of the correct dimensions, if necessary.) To remove every other point from both coordinate directions, the input-control file requires two passes for the adaption:

```
$namel twod=.t.,sub=1,rdsmax=4.0,rdsmin=.1,clam(1)=.005,jst=100,jend=1,nedge=1,indq=7 $
$namel sub=1,istep=.t.,noup=.t. $
```

Both passes use the SUB parameter, reducing the number of points to a 116×51 grid as shown in Fig. 20(c). The example shows that the SUB (as with the ADD parameter) can be used in conjunction with an adaption (as in the first pass) or simply as a method to reduce the number of points in the original grid (as in the second pass). It should be noted that if the two passes had been in reverse order, the input value of *jst* would need to be modified by the user to *jst=51*. To reduce points in a subset of the adaption domain, LSTSUB and LENDSUB should be used; however, in this example the default values of the entire adaption domain are appropriate.

(a)

(b)

(c)

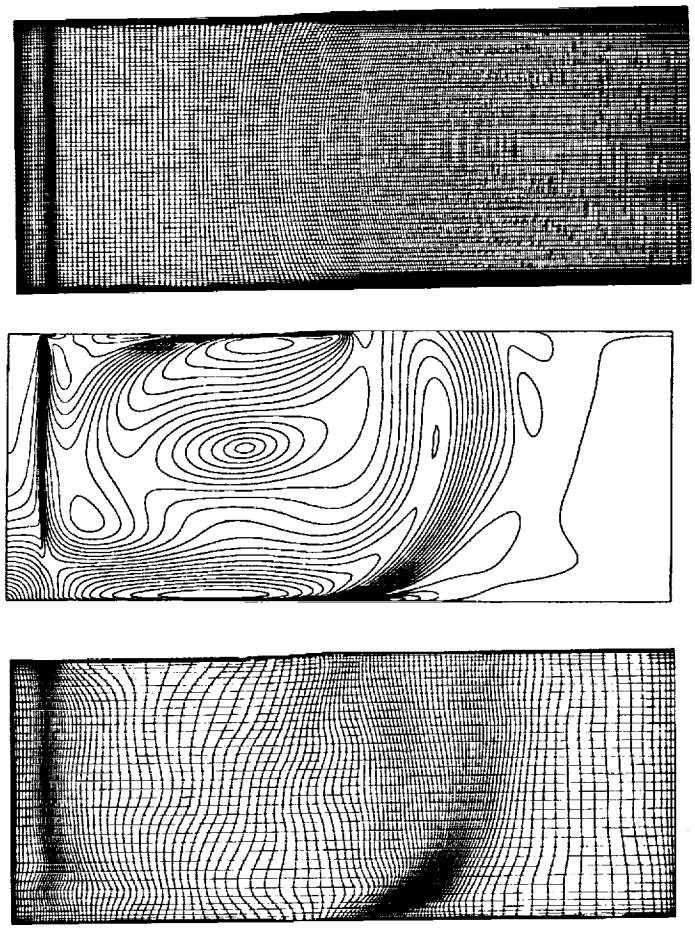

Figure 21. Subsonic impingement jet. (a) Initial grid; (b) initial mach contours; (c) adapted grid, with reduced points.

Case 6. Axisymmetric Plume Flow

This concluding 2-D case is presented to indicate the powerful effect of the adaptive grid process on the final flow-field solution. Figure 22(a) shows the initial grid and the reflected computed solution (in Mach contours) of an axisymmetric nozzle-plume flow described by Venkatapathy and Feiereisen (1989). This initial solution has not developed sufficiently to capture the final flow features: the outer shear layer, barrel shock, Mach stem, reflected shock, and the triple-point shear layer. Three iterations (through both adaption and flow solver) were made to produce the final adapted grid and solution shown in Fig. 22(b). The definition of the flow features is greatly improved. Figure 22(c) shows the accuracy of the final solution: the lower picture is a shadowgraph of the actual experiment and is almost mirrored by the computed solution.

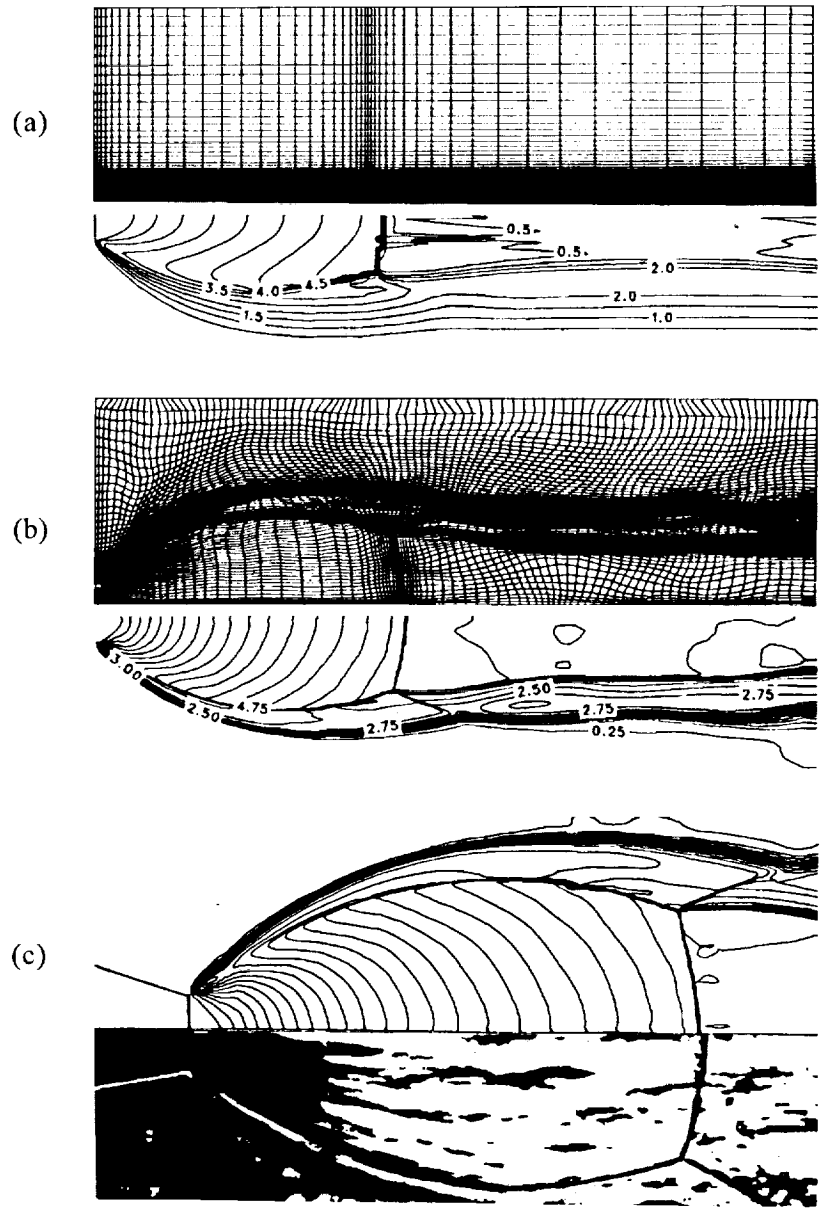

Figure 22. Axisymmetric plume flow. (a) Initial grid and flow solution (as Mach contours) showing underdeveloped flow features; (b) final adapted grid and flow solution after several iterations; (c) comparison of computed solution with experimental shadowgraph.

3.2 Three-Dimensional Examples

The adaption of 3-D grids is more complicated, not only because of the additional torsion control, but because of the increased choice of stepping and marching directions. For 2-D adaption, four choices of adaption are available: stepping in the *i* direction, the *j* direction, or both (in either order). Each of these options will produce a different adapted grid. For 3-D adaption, it is also necessary to choose a plane-stepping direction and, in theory, to consider up to three possible directional passes. In practice, it has been shown that a one-directional pass will frequently provide a sufficiently adapted grid, and that a two-directional pass is the maximum required. The following case shows the effect of single- and multi-directional passes.

Case 7. Tandem Fuel Injectors in a Supersonic Combustor (Rockwell Model)

Figure 23 represents a two-slot, tandem fuel injector arranged behind a backward facing step in a supersonic stream that models the Rockwell supersonic combustor. The fuel injection through the slot nozzle creates a complicated three-dimensional flow pattern. Since the outflow is supersonic, fuel injection normal to the main flow produces strong shock waves and streamwise separation in the vicinity of the slots. In addition, the backward facing step locally creates a subsonic flow ahead of the slots. In this example, grid adaption is limited to the region downstream of the backward facing step. The grid and initial solutions for the fuel injection problem were provided by J. Wang.

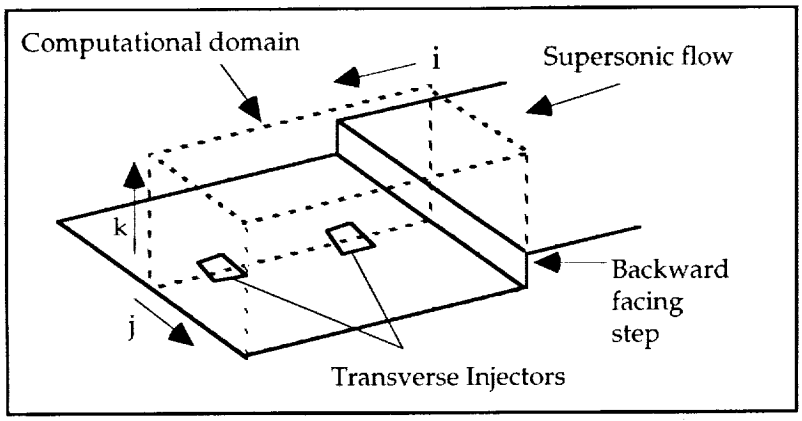

Figure 23. Staged transverse injectors in a supersonic combustor, showing computational domain.

Example 1: One adaption pass showing 3-D effect. This example is used to demonstrate how a one-directional adaption changes the grid-point distribution, not only on the chosen adaption plane, but also on the two cross planes. The initial grid ($80\times31\times61$) given in Fig. 24(a) shows three selected planes from Wang's initial grid, one in each of the coordinate directions, where *i=40*, *j=1* and *k=30*. For clarity, these planes are separated (but with their original orientations) and shown in Fig. 24(b). Figure 24(c) presents the corresponding Mach contours obtained from the solution on the initial grid by the flow-field code. The adaption is performed by redistributing the points on the first *j* plane and then marching in *j* planes (obtained by setting *ikplane=.t.*). Within each *j* plane, stepping is in the *i* direction (since *istep=.t.*) and the adaption is therefore performed along the *k* lines. The input parameter file used to create the adapted grids shown in Fig. 24(d) is:

$namel ikplane=.t.,istep=.t.,rdsmax=4.0,rdsmin=.2,indq=7,clam(1)=.0002,clam(2)=.0005,nedge=1 $

As requested, the *j* plane has clearly adapted to the Mach number gradients in the *k* direction, but so has the *i* plane, since this was the stepping direction within planes. However, the curvature of the planes themselves has not changed. The *k* plane shows a different effect: points have not moved within the plane, but only up and down (again, the *k* direction), thus changing the curvature of the plane. This example clearly shows that although the adaption is in only one direction, all planes are affected. It is thus quite possible for one pass to adequately adapt the grid for re-input to the flow-field code.

Example 2: Two-directional pass. This combustor problem is one that benefits from a two-directional pass. By looking at the constant *j* plane in Fig. 24(d), it can be seen that a second adaption to redistribute the *i* direction would be appropriate: this can be effected by adapting the *j* planes, stepping in *k* lines. The second set of adaption parameters are

$namel ikplane=.t.,kstep=.t.,rdsmax=4.0,rdsmin=.2,indq=7,clam(1)=.0002,clam(2)=.0005,nedge=1 $

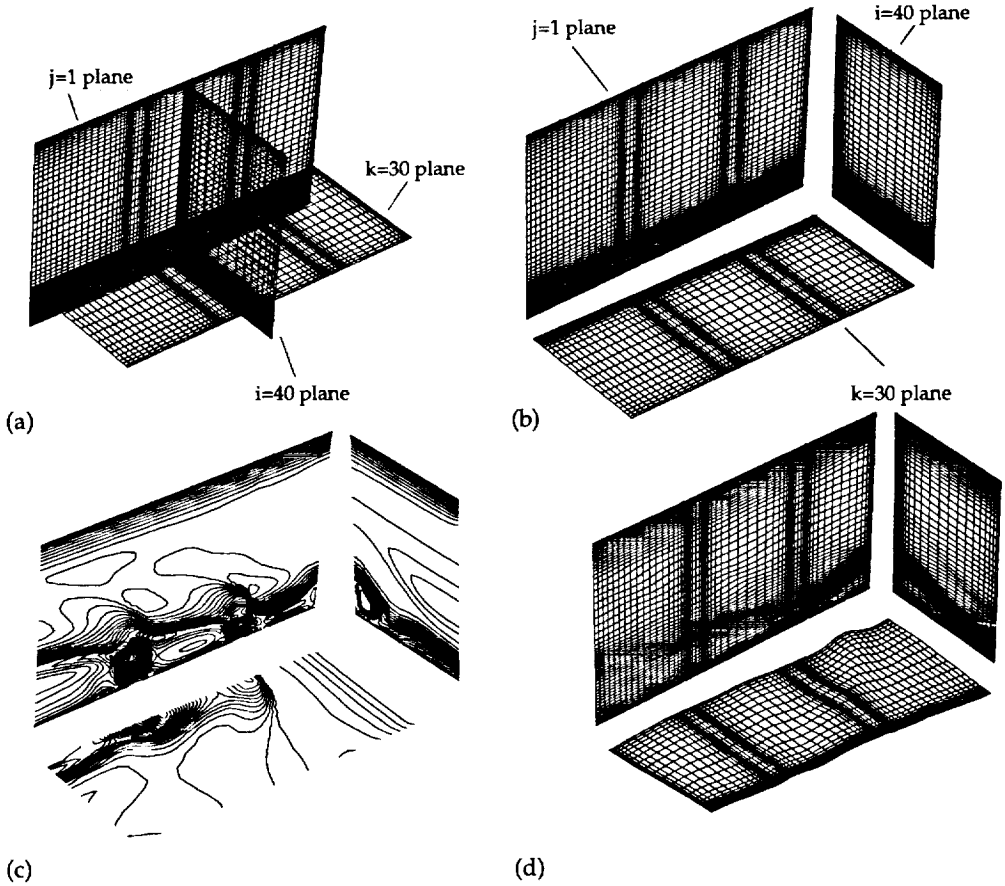

Figure 24. Three planes from initial grid. (a) Actual location, i=40,j=1,k=30; (b) same planes, separated for clarity; (c) initial Mach contours; (d) single direction adaption.

Figure 25(a) shows the result of performing this adaption "on top" of the adaption already shown in Fig. 24(d); points have now been redistributed in the *i* direction as well as the *k* direction. However, it should be noted that the *i* direction can also be adapted by marching in *j* within constant *k* planes by using:

$namel ijplane=.t.,jstep=.t.,rdsmax=4.0,rdsmin=.2,indq=7,clam(1)=.0002,clam(2)=.0005,nedge=1 $

This adapted grid is shown in Fig. 25(b) and shows a very similar redistribution on the *j* plane, even though this is not the adaption plane.

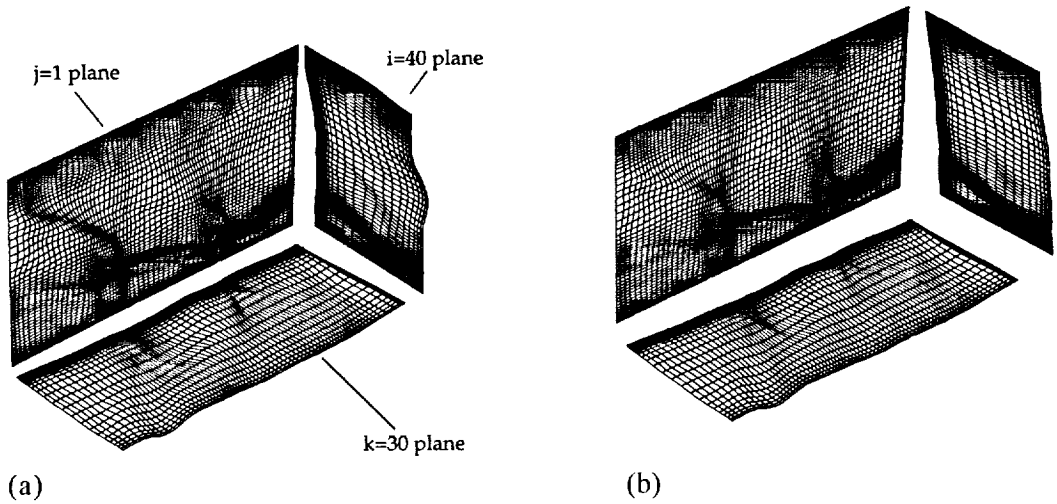

Figure 25. Two-directional adaption, using grid shown in Fig. 24(d) as input. (a) Adapting ik planes, stepping in k lines; (b) adapting ij planes, stepping in j.

Example 3: Torsion parameter between planes, λ^*. The 2-D examples show how the torsion parameter (λ) affects the adaption within a plane: it controls the magnitude of the torsion term and the larger the value of λ, the smoother the resulting grid, but with less grid redistribution. This effect is demonstrated in Case 1 in the 2-D section. Unfortunately, the "base" value of λ varies for each problem: the default value in the code is .01 but the user may have to change this value by as much as one, two, or even more orders of magnitude. λ^* is the analogous torsion parameter between planes, and acts in a similar manner to λ, but restricts the movement of points from plane to plane to maintain smoothness in the cross-planes. If $\lambda^* = 0$, there is no torsion control between planes, and planes will be adapted as independent entities. (In fact, setting $\lambda^* = 0$ is one way of handling periodic planes.) In the code, λ^* is defaulted to .0001 and this example illustrates the effect of this parameter.

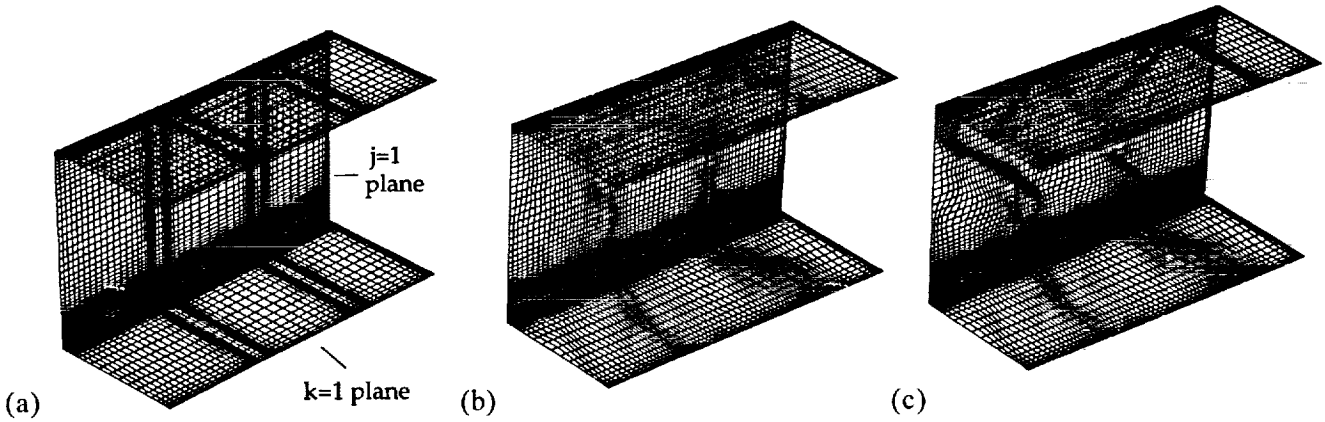

Figure 26. Effect of varying λ^*. (a) Initial grid showing j=1 plane; (b) adaption with clam(2)=.001; (c) adaption with clam(2)=.0001.

The same combustor-flow problem from Ex. 1 is used with different input control parameters. This time, we are adapting *k* planes and comparing the effect of λ^* on the crossing *j* planes. Figure 26(a) shows the initial distribution of two *k* planes (the upper and lower surfaces) joined by the first *j* plane. For this particular application, the first *k* plane would normally not be adapted, since the maintenance of its initial spacing is a high priority. However, to illustrate the adaption procedure, in this example the first plane is adapted. The input parameter file giving the result shown in Fig. 26(b) increases the value of λ^* to .001:

$namel ijplane=.t.,jstep=.t.,clam(1)=.001,clam(2)=.001,rdsmax=4.0,rdsmin=.2,nedge=1 $

This input requests marching in *k* planes (since *ijplane=.t.*) and stepping in the *j* direction within the *k* plane. The only parameter with any control of the *j* plane is λ^* (*clam(2)*) and the *j* plane shows some redistribution of points, but insufficient to reflect the flow features. Compare this plane to that in Fig. 26(c), where the adaption was performed with the identical input parameters except for a decrease in λ^* to .0001, i.e.,

$namel ijplane=.t.,jstep=.t.,clam(1)=.001,clam(2)=.0001,rdsmax=4.0,rdsmin=.2,nedge=1 $

The effect on the smoothing between planes is very evident and indicates the influence of the torsion parameter between planes.

Case 8. NASP 3-D Nozzle Simulation

Example 1. Two-directional adaption. Figure 27 shows the geometry of a wind tunnel experimental model for the National AeroSpace Plane (NASP) called the Single Expansion Ramp Nozzle (SERN). The model is inserted into a hypersonic test section with cold air injected at supersonic speed through the nozzle. Superimposed on the ramp section is an outline of the computational grid which is detailed in Fig. 28(a). This 3-D flow-field grid ($41 \times 60 \times 90$) defines the nozzle and after-body region of the model that is used in the computational experiments. Shown in the figure is the downstream outflow plane at *i=41*, the lower grid at *k=31* (part of which is the upper surface of the after-body ramp) and the symmetry plane at *j=1*. Additional patched grids are not shown, but there is a grid boundary that must be matched from *j=41* to *j=60* along the *k=31* plane with the grid that is stored in *k=1* to *k=30*, and also a matching

i=1 plane. Adaptions were performed from plane *i=2* to plane *i=41*, stepping in both *j* and *k* directions within the plane, using the initial solution shown as Mach contours in Fig. 28(b). This complex case utilizes many of the available input options, and the two-pass adaption parameters used to produce the adapted grid in Fig. 28(c) were:

$namel jkplane=.t.,kstep=.t.,rdsmax=2.5,rdsmin=.25,kst=32,kend=83,indq=7,nedge=1, clam(1)=.1,clam(2)=.01,mgsteps=4,mgpls=4,march=.t.,ct(1)=.75,save=.f. $

$namel jkplane=.t.,jstep=.t.,rdsmax=1.25,rdsmin=.25,kst=35,indq=7,nedge=2,mg1=8, clam(1)=.1,clam(2)=.01,mgpls=4,ct(1)=.75 $

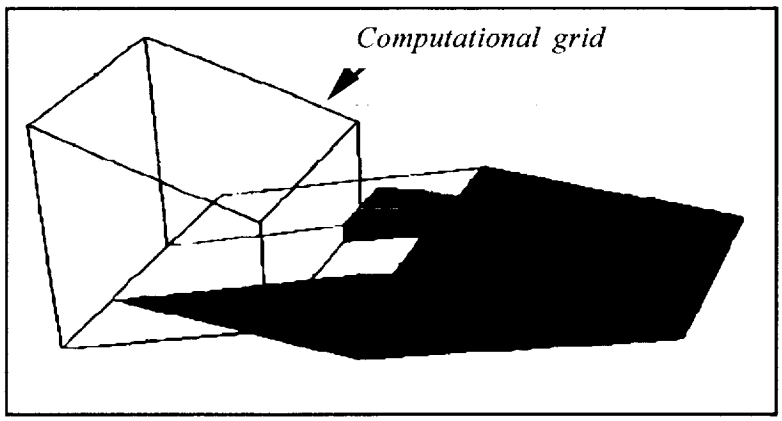

Figure 27. SERN experimental model and the computational space in the plume region.

The first set of parameters requests the adaption of each *i* plane (*jkplane=.t.*), stepping in the *k* direction (*kstep=.t.*) within the plane. Mach number (*indq=7*) is the adaption parameter. Adaption begins on line *k=31* with the merging technique invoked (*mgsteps=4*) within the plane. This will retain the *k=31* line to match the existing lower boundary and ensure that the adaption parameters are filtered for the next four adaption lines. Both torsion parameters and the directional parameter, *ct(1)*, are larger than their default values since a previous test run showed that too much adaption (and thus loss of smoothness) occurred using the default values. Another parameter used in this example is MARCH. This is invoked from line *kend=83*: after this line, the flow has no gradient features and is mostly numerical 'noise,' and MARCH simply presents a more attractive grid. Since the adaption algorithm normalizes the weighting function, 'noise' can produce unnecessary redistribution. Due to adjacent grids, it is not appropriate to adapt the *i=1* plane, thus a smooth transition is required between the first plane and subsequent planes. This is controlled by *mgpls=4*: the default start plane of *ist=1* was not adapted and the next three planes were more gradually adapted than they would have been if MGPLS was not requested. Example 2 of this case gives a detailed account of this plane-merging process. The SAVE parameter was set to .false. to prevent the output datasets from being written. This should be done only when more than one adaption pass is made in the same computer run.

The second adaption set requests adaption of the same *i* planes, but stepping in the *j* direction within planes. The first adaption point is *k=35*. This was chosen to retain the first few points in the boundary layer. In addition, the parameters *nedge=2* and *mg1=8* are used to filter this boundary layer spacing into the next eight grid points on the line. The value of *nedge* was changed from 1 to 2 for this pass because there was no need to maintain any spacing at the outer edge, where no gradients are found: these points are better used within the body of the grid. However, the adaption still left an unnecessary number of points in this outer region, so a final pass was made:

$namel jkplane=.t.,jstep=.t.,noup=.t.,sub=1,lstsub=82,add=1,lstadd=51,lendadd=55 $

Here, no adaption is performed (*noup=.t.*) and four lines are moved by using the SUB and ADD parameters. The final adapted grid is given in Fig. 28(c). This was input to the flow solver, producing the final solution shown in Fig. 28(d).

It should be noted that this problem is a multigrid problem and that adaption of grids with matching planes can be facilitated by using the export and import capabilities described elsewhere.

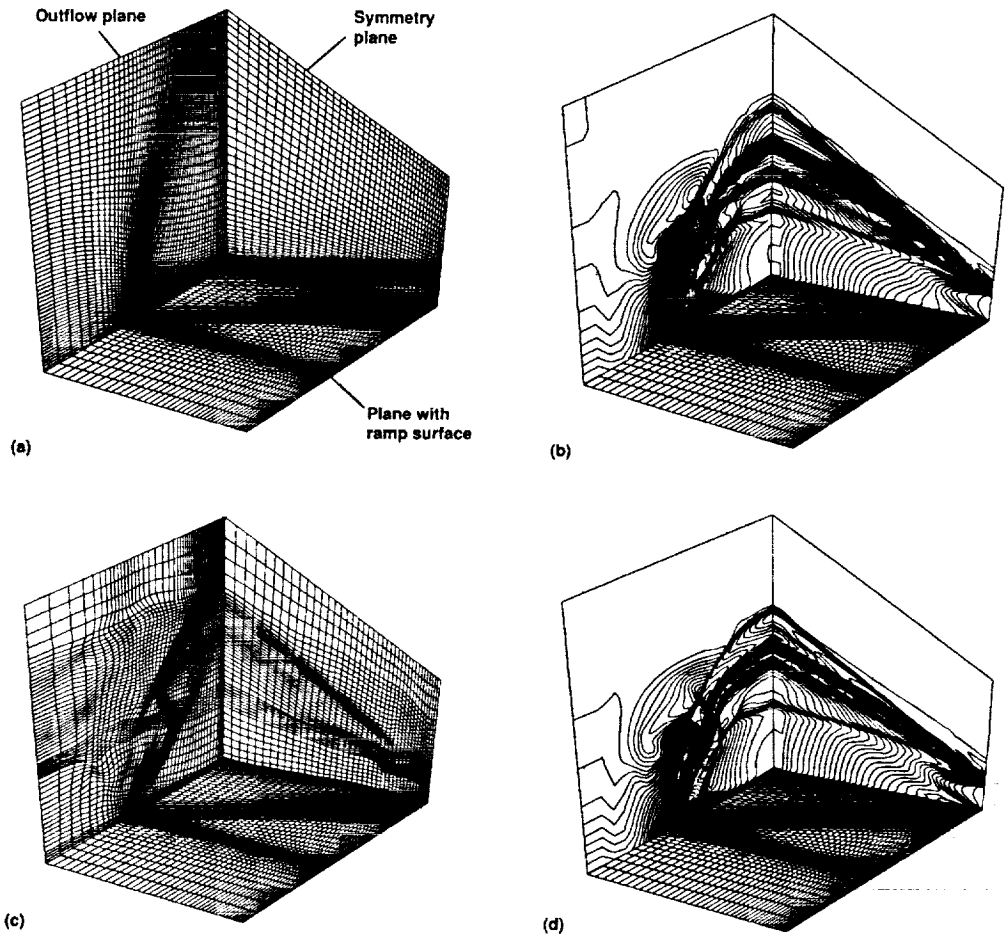

Figure 28. NASP nozzle plume flow. (a) Initial grid; (b) initial Mach contours; (c) adapted grid; (d) final solution using adapted grid.

Example 2: Merging from non-adapted planes to adapted planes. The nature of three-dimensional problems frequently necessitates the maintenance of boundaries: either for multiple grids or to preserve solid geometry walls and even for maintaining boundary layers. For the NASP nozzle case described here, the plane at $i=1$ must match another plane on the adjacent (not shown) grid. (Note that this plane may have received data via the export/import process.) There is also a dense grid spacing around the nozzle exit for defining the boundary layer region that should also be retained. The way to handle these two situations is to not adapt the first plane, but to adapt the contiguous planes in a merging manner. This merging is needed to create a smooth transition in the cross plane, not in the adaption plane itself (which is handled by MGSTEPS).

Figure 29(a) shows the initial $i=1$ (and thus the identical $i=2$ plane) for the 3-D grid shown in Fig. 28, and Fig. 29(b) contains the corresponding pressure gradients for the $i=2$ plane. If the input adaption parameter $ist=2$ is used, with no merging process invoked (i.e., the first plane is ignored) the resultant adapted $i=2$ plane is shown in Fig. 29(c). Although this grid is smooth and nicely adapted to the flow, the cross plane, j, is not (compare Fig. 29(c) with Fig. 29(a)). In addition, points have been drastically pulled away from the nozzle region. By invoking the MGPLS option, this cross-plane unevenness can be dampened. The technique is analogous to that used by MGSTEPS: both λ^* and C_l^* are modified to increase the restraint of movement of points between planes. These values will return to their original input values within a certain number of planes, as requested by MGPLS. The following is the input parameter file used to create the adapted grid ($i=2$ plane) shown in Fig. 29(d):

```
$name1 jkplane=.t.,jstep=.t.,rdsmax=1.25,rdsmin=.25,kst=35,indq=6,nedge=2,mg1=8,
       clam(1)=.1,clam(2)=.01,ct(1)=.75,mgpls=4 $
```

The difference between Figs. 29(c) and (d) is striking: the boundary layer spacing is now maintained and the adaption process is just beginning. Although *mgpls=4* in this example, setting *mgpls=1* will also have considerable effect. By setting MGPLS at all, a request is made to not adapt the first requested plane (in this case, *ist=1*, by default) but to use the grid on that plane as a control for the subsequent planes. Thus, *mgpls=1, ist=1* is not the same as setting *ist=2*.

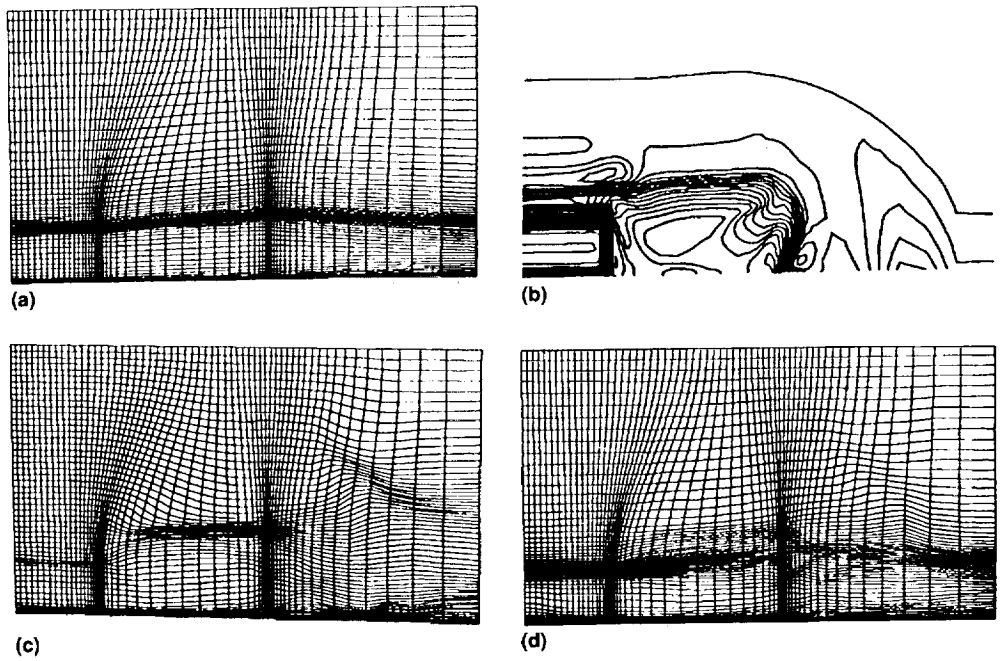

Figure 29. Merging planes using mgpls. (a) Initial grid at i=1 (and i=2); (b) initial pressure contours; (c) adapted grid at i=2 with ist=2; (d) adapted grid at i=2 with ist=1 and mgpls=4.

Case 9. Aeroassist Flight Experiment (AFE) Vehicle

This example is the hypersonic, non-equilibrium flow around the forebody of the aeroassisted flight experiment (AFE) vehicle. In Fig. 30(a) the initial grid configuration ($35\times23\times49$) is shown around the body in the form of j planes 2 and 22, and the outflow plane at $i=35$. The Mach contours seen in Fig. 30(b) were computed by the flow-field solver of Palmer (1990) using the non-adapted grid. This grid was then adapted with respect to the Mach contours, and the resulting grid is seen in Fig. 30(c). The redistribution of points within the blunt-body shock region is clearly shown. Due to the smooth shape of the shock, adaption was performed only in one direction: marching in j planes and stepping in the i direction within planes. The input data set used to create the adapted grid in Fig. 30(c) was

```
$namel  lnsing=1,ikplane=.t.,istep=.t.,indq=7,jst=2,jend=22,rdsmax=4.0,rdsmin=.1,
        kst=15,orthe=.f.,nedge=1,clam(1)=.001 $
```

This is the first example to use the LNSING parameter. It is set since all j planes emanate from a single line at $i=1$. (This forebody singular line is used in grid mapping.) However, the dataset still contains the grid points for line $i=1$ as if it were a separate line in each plane. *lnsing=1* ensures that the first line in the first plane is adapted, and that all $i=1$ lines on subsequent planes are set identical to this adapted line. The parameter *orthe* was set to *.false.*, which removes the orthogonality constraint at the outflow line of $i=35$. The default value of *.true.* would have created a false turning of the adaption at this location. This adapted grid was then re-input to Palmer's flow-solver which showed a considerably sharper blunt-body shock feature (1990).

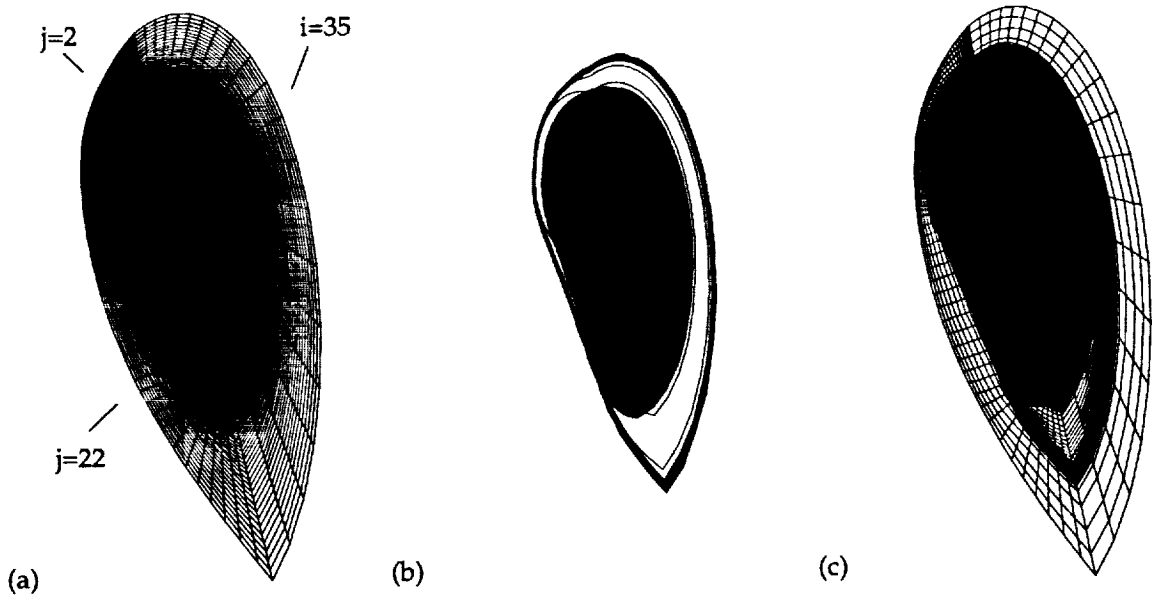

Figure 30. Hypersonic forebody flow (AFE). (a) Initial grid around the forebody; (b) initial Mach contours; (c) adapted grid.

3.3 Multiple-Grid Examples

Section 2.4.3 describes some of the complexities of multiple grids. The first case here reproduces the stylized example shown in Fig. 11 to illustrate the transfer of planes from one grid to another. The second case is a generic NASP configuration (Davies and Deiwert, 1993).

Case 10. A Simple 3-D Grid

Example 1: Single plane transfer. Figure 31 is a reproduction of Fig. 11 from Section 1. This example shows a matching common plane that is stored separately in each grid, but contains identical data. We will define the size of each grid as (i,j,k) of $(40\times15\times30)$. If the grids had been separated as single grids, as shown in Fig. 31(b), the only choice would be to start the adaption process at the common plane and to march in opposite directions. This would provide identical adaption points for the common plane. For some problems this could be quite acceptable, but marching in the same direction more often produces the preferred result.

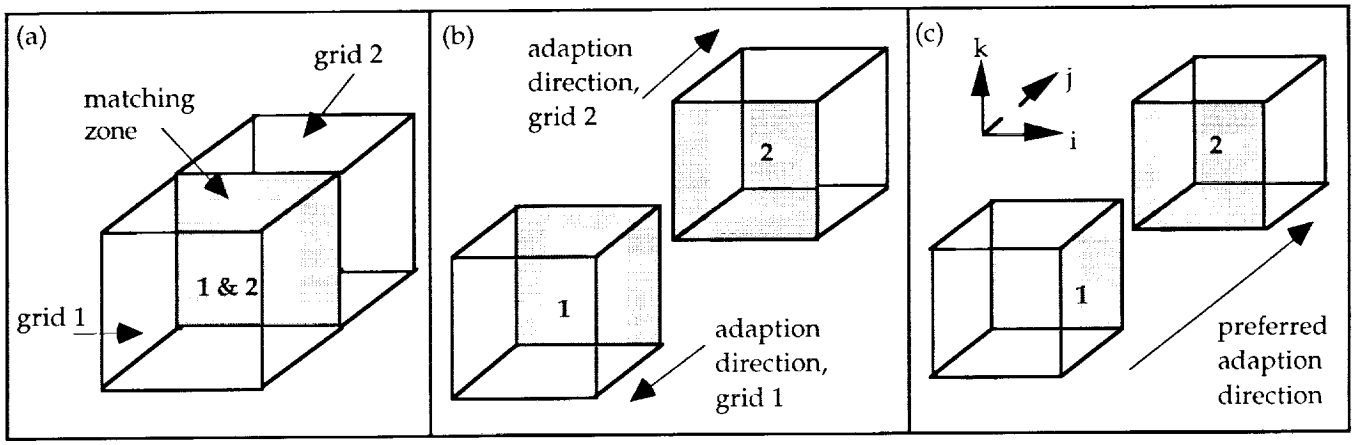

Figure 31. Reproduction of figure 11.

To adapt multizoned grids as shown in Fig. 31(c), a feature is available to transfer data from one grid to another. The adaption process in Fig. 31(c) is to:

(1) adapt grid 1 up to the common plane ($j=15$);
(2) transfer data from the last plane in grid 1 to the first plane in grid 2; and
(3) adapt grid 2, using the merging planes option (i.e., *mgpls* nonzero) for the first plane.

This sequence is described by the following input control list:

```
$namel mgrid=1,ikplane=.t. $
$namel export=.t.,mgrid=1,ikplane=.t.,mplane=15 $
$namel import=.t.,mgrid=2,ikplane=.t.,mplane=1 $
$namel mgrid=2,mgpls=2,ikplane=.t. $
```

If only a subset of plane 15, grid 1 was to be transferred to grid 2 (as shown in Fig. 32), then the range would be described by entering values for parameters *is,ie,ks,ke* (i.e., those relating to the *ikplane*) for both grids by

```
$namel export=.t.,mgrid=1,ikplane=.t.,mplane=15,is=30 $
$namel import=.t.,mgrid=2,ikplane=.t.,mplane=1,ie=10 $
```

In this case, points $30 \le i \le 40$ and all k points from the 15th *ikplane* in grid 1 are transferred to points $1 \le i \le 10$ in the first plane of grid 2. (Note that only *is=30* on the export list and *ie=10* on the import list are required since the other range values are the same as the default values.)

Note for 2-D files. The same concept holds for 2-D files but care must be taken since only a line is being transferred. Because only one plane exists, *ijplane=.t.* and *mplane=1*. Only *is,ie,js,je* can be used and one of these two sets must have equal first and last values. Finally, if the export card is the first card in the adaption set, include *twod=.t.*

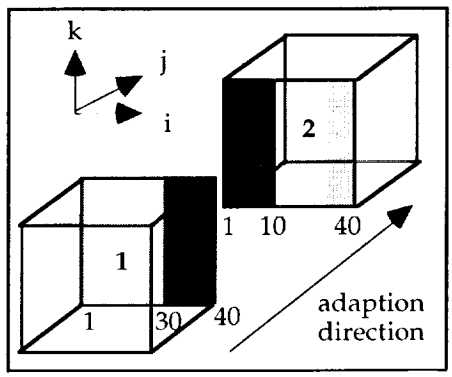

Figure 32. Matching subsets.

Example 2: Multiple transfers. It is possible to handle more than one transfer within the same run of the code. In the example shown in Fig. 32, it is probable that the lighter shaded domain of plane 15, grid 1, matches to a third grid, not shown. For this example, the two sets of transfers can be entered sequentially:

```
$namel export=.t.,mgrid=1,ikplane=.t.,mplane=15,is=30 $
$namel import=.t.,mgrid=2,ikplane=.t.,mplane=1,ie=10 $
$namel export=.t.,mgrid=1,ikplane=.t.,mplane=15,ie=30 $
$namel import=.t.,mgrid=3,.ikplane=.t.,mplane=1 $
```

Case 11. Generic NASP Configuration

This case is the result of NASP Government Work Package #20, by Davies and Deiwert (1993). A two-part multiple grid defines a generic NASP vehicle configuration: one grid ($35\times59\times51$) for the nose region and one grid (also $35\times59\times51$) encompassing the remainder of the body. Figure 33(a) shows the vehicle surface and three cross planes (*jk* planes), and Fig. 33(b) shows the corresponding Mach contours that were used as the adaption function. The i direction is through the vehicle centerline, j steps away from the body and k marches around the body from the lower surface. Adaption begins at the nose and marches plane by plane towards the end of the body, transferring data at the matching plane (*grid 1, jkplane 35* to *grid 2, jkplane 1)*. The input control file used to produce the adapted grid shown in Fig. 33(c) is

```
$namel mgrid=1,jkplane=.t.,ist=2,kstep=.t.,kst=51,kend=1,jst=15,indq=7,nedge=1,
       rdsmin=.1,rdsmax=3.0,clam(1)=.001,ct(2)=.8,orthe(1)=.f.,orthe(2)=.f. $
$namel export=.t.,mgrid=1,jkplane=.t.,mplane=35 $
$namel import=.t.,mgrid=2,jkplane=.t.,mplane=1 $
$namel mgrid=2,jkplane=.t.,kstep=.t.,kst=51,kend=1,indq=7,mgpls=4,nedge=1,
       rdsmin=.1,rdsmax=3.0,clam(1)=.001,ct(2)=.8,orths(2)=.f.,orthe(1)=.f.,orthe(2)=.f. $
```

Some of these adaption parameters need explaining: *ist=2* is the starting plane since the nose at *ist=1* is a plane collapsed to a single point; *kst=51* and *kend=1* produces the best adaption direction for *kstep* (compared to the result using the defaults *kst=1* and *kend=51*); *jst=15* preserves the boundary layer; and *ct(2)=.8* (more straightness for the between-planes torsion parameter) produces a smoother adaption when looking at the other planar directions. The export and import processes are straightforward. For the

adaption of grid 2, *mgpls=4* and *orths(2)=.f.* both contribute to maintaining continuity across the matching plane.

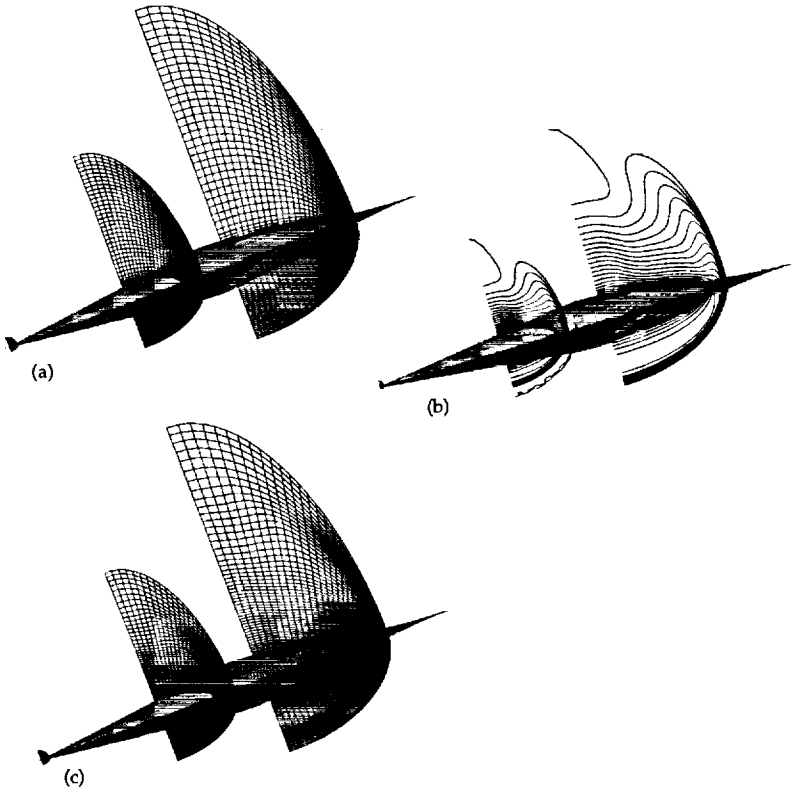

Figure 33. Generic NASP vehicle configuration. (a) Initial grid; (b) initial Mach contours; (c) adapted grid.

3.4 Recluster and Boundary Movement Examples

Reclustering is an alternative to adaption. It was initially added to give the user a second option for point redistribution after the outer boundary was moved. (The first option was to proportion the new grid spacing to the old grid spacing.) However, the same reclustering algorithm proved to be a useful tool to redistribute the grid points without any boundary movement. An example would be to convert a grid developed for an inviscid calculation into one appropriate for viscous flows. The first example below demonstrates the reclustering technique without boundary movement; further examples demonstrate boundary movement options.

Case 12. Space Shuttle

A single grid around the Space Shuttle, shown in Fig. 34(a), is used for these examples. The *i* direction runs from the nose to the tail, and the *k* direction is from the body surface to the outer boundary, thus the mesh seen in Fig. 34 is an *ikplane*. This initial grid is equally spaced to more clearly demonstrate the reclustering process. Note that SAGE can transform any grid into an equally spaced grid by using *rdsmax=1.0, rdsmin=1.0, clam(1)=.0, clam(2)=.0.*

Example 1. Reclustering entire domain with no boundary movement. To create a grid with the stretched spacing shown in Fig. 34b, the following input dataset was used:

$name1 lnsing=1,ikplane=.t.,istep=.t.,reclust=1,mvbound=0,noq=.t.,dsw=.0001,$

The parameter *lnsing* is set because of the singular line at the nose, and *mvbound=0* signifies no boundary movement. No Q file is needed to recluster the grids, so *noq* is set to true (however, this implies that any existing Q file will not be interpolated onto the new grid points). Setting *reclust=1* is the same as

reclust=kmax (*ikplane* and *istep* determine which lines we are reclustering, in this case, lines of constant *k*), hence all the points will be reclustered. The value of *dsw* (.0001) specifies the actual size of the first spacing off the wall, not a normalized value.

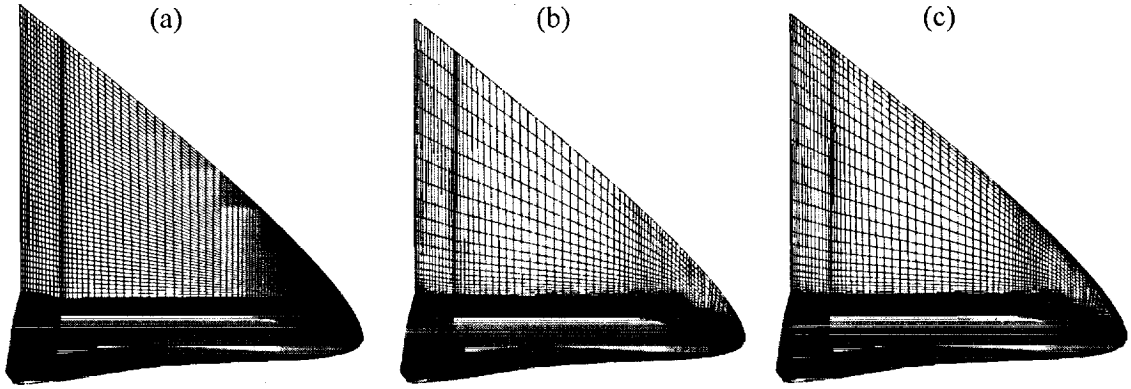

Figure 34. Space Shuttle grid. (a) Initial, equally spaced grid; (b) new grid, reclust=1; (c) new grid with reclust=1 and dse=1.0.

Figures 34(b) and 34(c) show the effect of the reclustering. Points are tightly spaced next to the vehicle surface for both cases. In Fig. 34(b), the outer Vinokur parameter, *dse*, contains the default value of 5.0 (which sets the outer spacing to five times the average spacing); however Fig. 34(c) was created by inputting *dse*=1.0, which retains a more dense spacing at the outer boundary.

This grid is an example where the outer boundary is close to the vehicle surface at the nose and then rapidly expands away from the vehicle surface downstream. In this situation, the absolute size of the first wall spacing, $\Delta s_{i=1}$, at the nose may not be appropriate for $\Delta s_{i=i\,max}$. In addition, such tight spacing at the nose can create difficult conditions for the Vinokur solver further along the body, producing an unsmooth grid. This is clearly shown in Fig. 34(d) where the value of .00001 has been used for the wall spacing all along the body.

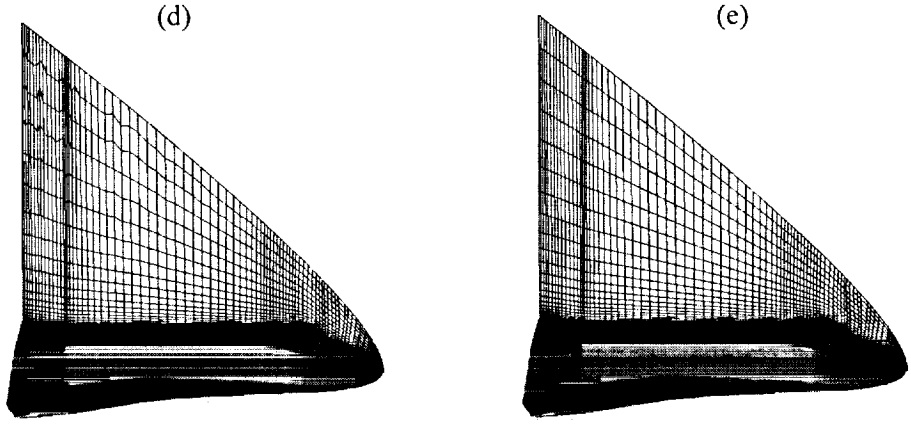

Figure 34 concluded. (d) Constant wall spacing, dsw=.00001; (e) expanding wall spacing, dsw=999, dsw1=.00001, dsw2=.0001.

To create the smoother grid seen in Fig. 34(e), the user can set *dsw=999.*, and input two values for the wall spacing, one at the nose (*dsw1 =.00001*) and a larger value at the tail (*dsw2 =.0001*), by using the input control file:

$namel lnsing=1,ikplane=.t.,istep=.t.,reclust=1,mvbound=0,noq=.t., dsw=.999.,dsw1=.00001,dsw2=.0001 $

If reclustering is required within a subset of the grid, as for example, within the boundary layer, the input control file would be the same, except for *reclust* , which would now be set equal to the number of points to recluster, where *reclust<kmax.*

Example 2: Percentage boundary movement. To move the outer boundary by a percentage of its original location, *mvbound* is set equal to the percentage required. Figure 35(a) shows the same original grid as Fig. 34(a). The input file

$namel lnsing=1,ikplane=.t.,istep=.t.,reclust=0,mvbound=–20,noq=.t., $

produces the grid shown in Fig. 35(b): the negative sign on *mvbound* pulls in the outer boundary by 20%.

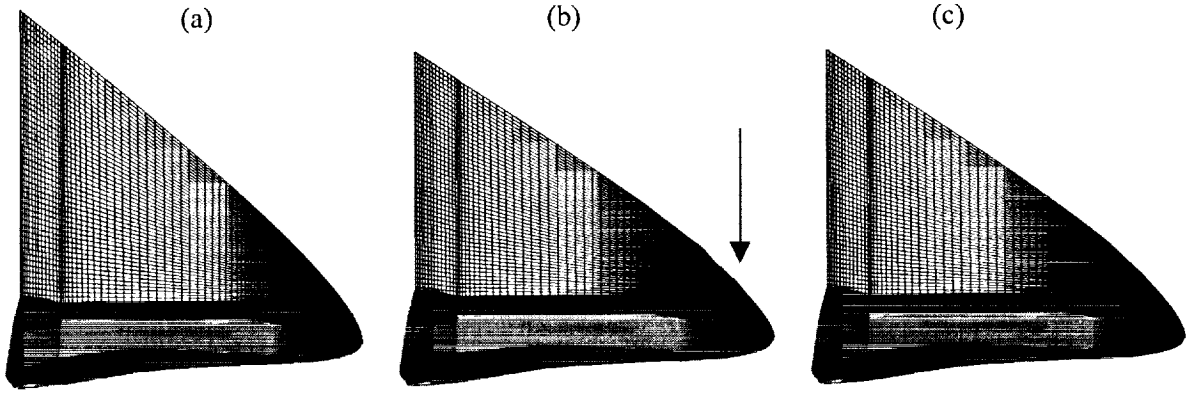

Figure 35. Percentage move in outer boundary. (a) Original grid; (b) mvbound=–20; (c) mvbound=–20, nsm=0.

If *mvbound=20* had been used, then the boundary would have moved out by 20%. This example also provides a glimpse into the smoothing parameter *nsm*, which has a default value of 10: the smoothing has caused a blip (shown by the arrow) at the change of boundary length over the shuttle window. The case was rerun, adding *nsm=0* to the same input file above, giving the result seen in Fig. 35(c). In both cases, *reclust=0* provides a new grid distribution proportional to the original spacing.

Example 3: Outer boundary movement parallel to shock. This example demonstrates moving the outer boundary to match a shock. Figure 36(a) is the initial grid and the computed density contours are shown in Fig. 36(b). The following input dataset was used to produce the grid shown in Fig. 36(c):

$namel lnsing=1.ikplane=.t.,istep=.t.,reclust=1,dsw=.001,mvbound=999,indq=1 $

When *mvbound=999*, the code is told to use the *q* variable of record (in this case *indq=1*, density) to establish the new location of the outer boundary. The code will search from the outer edge until a jump in density gradient is found. In the example, the boundary is moved inwards, but it is possible for a boundary to move outwards if the shock crosses through the outer edge. However, if the outward movement is not satisfactory, one technique is to move the boundary out by *mvbound=+%* and then re-run with a *mvbound=999*. This will ensure that the entire shock is captured downstream and points are not wasted in the nose region.

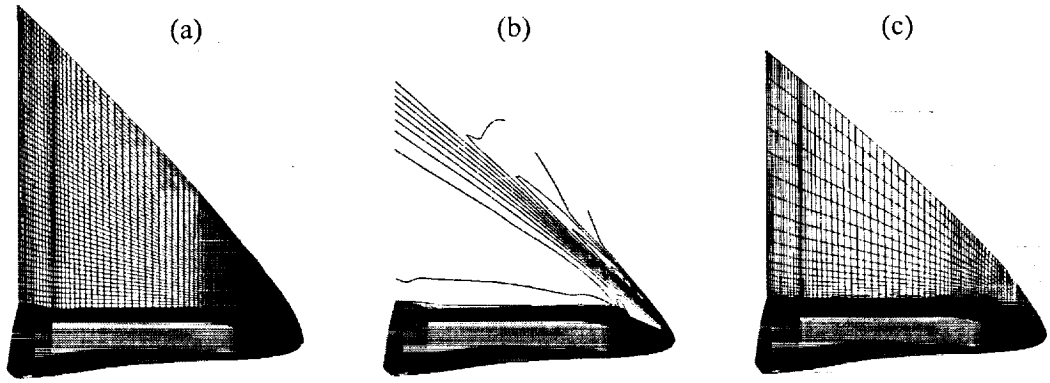

Figure 36. Outer boundary movement. (a) Initial grid; (b) density contours; (c) new grid, with outer boundary moved inwards and points reclustered.

There is another option to move the boundary to match a contour line. If *mvbound > 1000.0*, then the boundary will run parallel to a particular contour line whose value is *mvbound–1000.0*. Choosing the variable (i.e., *indq*) and the value of the variable needs care. For example, consider the following input:

$namel lnsing=1,ikplane=.t.,istep=.t.,reclust=1,dsw=.001,mvbound=1400.0,indq=5 $

Note that the fifth Q variable in this example contains temperature. The temperature contours and the resulting grid are shown in Figs. 36(d) and (e). The value of *mvbound=1400.0* requests the outer boundary to run parallel and close to the 400° contour line. Care must be taken to ensure that the chosen value exists along all lines to prevent an unacceptable result.

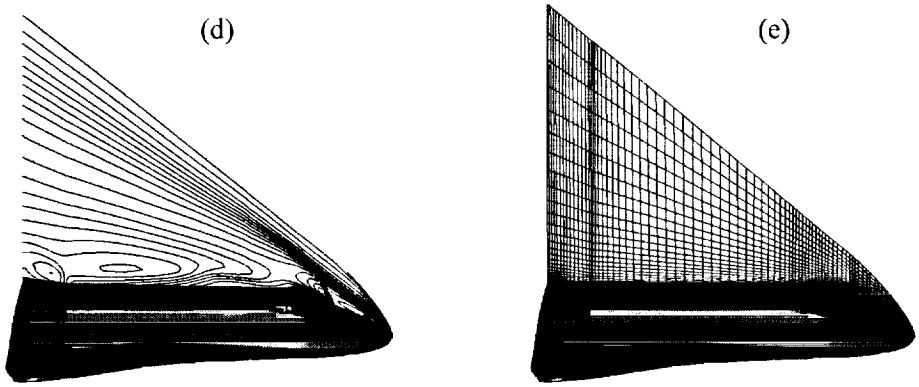

Figure 36 concluded. (d) Temperature contours; (e) outer boundary line follows the $T = 400^{\circ}$ contour.

4. REFERENCES

Davies, C.B.; and Deiwert, G.S.: Generic Grid Adaption, NASP Contractor Report 1143, February 1993.

Davies, C.B.; and Venkatapathy, E.: Application of a Solution Adaptive Grid Scheme, SAGE, to Complex Three-Dimensional Flows, AIAA Paper 91-1594, June 1991.

Davies, C.B.; and Venkatapathy, E.: A Simplified Self-Adaptive Grid Method, SAGE, NASA TM-102198, October 1989.

Davies, C.B.; and Venkatapathy, E.: The Multidimensional Self-Adaptive Grid Code, SAGEv2, NASA TM-110350, April 1995.

Nakahashi, K.; and Deiwert, G.S.: A Self-Adaptive-Grid Method with Application to Airfoil Flows, AIAA Paper 85-1525, July 1985.

Palmer, G.: Enhanced Thermochemical Nonequilibrium Computations of Flow Around the Aeroassist Flight Experiment Vehicle, AIAA Paper 90-1702, 1990.

Venkatapathy, E.; and Feiereisen, W.J.: Computational Studies of Hard-Body and 3-D Effects in Plume Flows, AIAA Paper 89-0129, 1989.

Vinokur, M.: On One-dimensional Stretching Functions for Finite-Difference Calculations, Journal of Computational Physics, Vol. 50, No. 2, May 1983.

Walatka, P.P.; Buning, P.G.; Pierce, L.; and Elson, P.A.: PLOT3D User's Manual, NASA TM-101067, March 1990.

5. SAGEv2B, THE BLANKING VERSION

This section describes the blanking version of SAGE. SAGEv2 was modified to take into account the BLANK feature available in PLOT3D, and was renamed SAGEv2B. The current SAGEv3 described in this document does not accept the input parameter BLANK described below. Users interested in this option should contact the authors for SAGEv2B. When time permits and/or if there is sufficient demand, this feature will be added to the latest version. However, the following description will remain valid.

5.1 Algorithm for Blanked Overset Grids

As mentioned in Section 1.10.1, an overset blanking grid system can produce complicated grid structures. Grids in the same system can have different functions. For example, a fine, detailed grid may be created around each physical object, along with a larger, less dense grid that overlays the entire domain. In fact, several layers of nested grids may be appropriate. Nevertheless, as with any grid generation method, grids are still created around the physical objects with no a priori knowledge of the solution. Since the accuracy of the flow solution is dependent on the grid, the need for an adaptive grid scheme is still important.

One obvious option is to ignore the blanking flag and adapt the grid using whatever data is available in the blanked regions. However, in most cases these values are not relevant to the solution. Therefore, a change in the algorithm and some careful coding is required.

The basic adaption equation is given by Eq. (7):

$$\omega_i \Delta s_i - \omega_{i-1} \Delta s_{i-1} + f(\tau_i) + f(\psi_i) = 0$$

where s is the streamwise length along the adaption line, ω is the tension force between points on a line, τ is the within-plane torsion force, and ψ is the between-plane torsion force. The τ torsion force is a function of the torsion vector defined in Eq. (16), where the constants t_n and C_t may change from line to line but are constant along a line, and hence are defined as single variables. When blanked regions are involved, there will be active points along a j line that may be blanked (i.e., inactive) on the $j-1$ line. For the straightness vector $\vec{e}$, data is also required from the $j-2$ line. A similar situation exists for the between-plane torsion vector, $\hat{t}_i^*$. A simple method was chosen to handle the computation of the torsion vectors when blanked points are found. First, the values of C_t and t_n were made dependent on i. If the point *(i,j,k)* exists, but *(i,j−1,k)* is blanked, then $t_{n_i} = 1$ and $C_{t_i} = 0$ will place all the torsion force onto the orthogonal vector $\hat{b}_i$ acting on the j line. If the points *(i,j,k)* and *(i,j−1,k)* exist, but *(i,j−2,k)* is blanked, then the straightness vector is amended to include the $j-1$ point only. The same changes are made to the coefficients of the ψ vector, depending on the existence of the *(i,j,k−1)* and *(i,j,k−2)* points.

Another significant difference is the number of points on a line. With the blanking feature invoked, the number of i points along a j line can differ line by line, varying from the user's input domain. The user specifies the maximum value of *(ist,iend)*, *(jst,jend)*, and *(kst,kend)* for the adaption domain; the code will recompute the start and end points of the adaption line for each step, within the user's specified limits. When blanking occurs in the middle of a line, the two segments must be adapted separately.

Special handling occurs at boundaries. For edge boundaries, continuity is maintained between adaption and physical boundaries. A question occurs along a boundary created by a blanked region: is this edge to be externally influenced or not? It is currently assumed that edge control will not occur at a blanked boundary unless the blanked boundary point is closer to the outer domain edge than the value of the corresponding MG value. For marching boundaries in blanked applications, the code determines the start and end points along the last adapted line (either *jend,k* or *j,kend*, depending on MARCH or MARCHPL). If a line to be interpolated starts (and/or ends) within the range of the last line, then only the appropriate points will be used from the last line. If there are more points along the interpolation line, only the available range from the last line will be used: the remaining points will retain their initial locations.

The addition or subtraction of points has been handled very simply. On request, points will be added or removed over the entire adaption domain, regardless of blanked regions. The value placed in the blanking array will reflect the proper position of each point, indicating whether it is in the blanked region or not.

The proportionality coefficient, λ, controls the magnitude of the torsion parameter. When a situation occurs that drastically increases the number of points in a line, the value of λ, although appropriate for the overall adaption, may permit too much movement when integrating the blanked region into a non-blanked region. For this reason, λ is locally magnified to prevent excessive point movement.

5.2 Execution of SAGEv2B

SAGEv2B is the only version of SAGE that recognizes the blanking option that is found in PLOT3D grid files as an additional variable, called IBL. The changes to the adaption algorithm to handle blanked areas are discussed above. (Note: If IBL is defined in the PLOT3D input grid file, SAGEv3 can be used but the blanking will be ignored and the output grid file will not contain IBL.)

Invoking the blanking option is extremely easy: the variable *blank=.t.* should be added to the input parameter file. However, defining the adaption domain and the marching directions requires much more care. With the blanking feature invoked, the number of *i* points along a *j* line can differ line by line, varying from the user's input domain. The user must specify the maximum value of *(ist,iend)*, *(jst,jend)*, and *(kst,kend)* for the adaption domain; the code will re-compute the start and end points of the adaption line for each step, within the user's specified limits. If the blanked region is in the middle of the grid, then another strategy must be developed. One option is to divide the adaption domain into two. Unfortunately this fixes one of the lines or creates a discontinuous grid. Performing two adaptions, sweeping in opposite directions, will provide a better result. The examples section will clarify this.

5.2.1 User Input Parameter

There is only one new input parameter:

BLANK (FALSE) set to .true. to read PLOT3D IBL variable

BLANK The grid file in PLOT3D has an additional optional variable. Following the X,Y,Z arrays is an array called IBL. IBL is the same dimension as the grid and contains an integer value that describes the status of the grid point. The standard nomenclature of IBL=1 implies the grid point is active, and IBL=0 signifies an inactive point. Some users may put other values in IBL, for example IBL=2 to represent a border point. This code assumes that if IBL=0, the point is not considered; otherwise, it is assumed to be part of the grid and will be counted as an active point on a line. The input values of IST, IEND, etc., that describe the active adaption domain must contain the maximum required domain; the code will determine the active domain for each line based on the values of IBL and these maxima.

5.3 Subroutines and Variables

Several previously defined subroutines are amended. For example, all the I/O routines are changed to handle the IBL variable (including reading, writing, swapping etc.) Also, variables such as NIPTS, MG1, etc., that were constants but now vary are renamed so they can be retrieved, and variables that become arrays (such at CT) are defined. Because of its expanded size, the routine *INITIAL* is split into two: *INITIAL* and *GETDATA*. In addition, three new subroutines were created.

*BLTST(IB,JB,KB,**BLANKIT**)*

This routine tests to see if any points are blanked around an existing point *(ib,jb,kb)*, for example, a 'hanging' point. If so, this point is also treated as blanked since there is no connectivity for adaption. The variable BLANKIT is therefore set to true.

LNPTS(J,K)

This routine computes the number of active points on a line for adaption. Blanked points are not used. Also the edge merge parameters (MG1 and MG2) are amended if blanking occurs within the line edge.

TORBL(J,K)

This routine amends the torsion coefficients C_t and t_n for *(i,j,k)* to reflect the status of adjacent points. If the point at *(i,j−1,k)* is blanked, then within-line coefficients are changed to $C_{t_i} = 0$ and $t_{n_i} = 1$. If *(i,j,k−1)* is blanked, then the within-plane coefficients are changed to $C^*_{t_i} = 0$ and $t^*_{n_i} = 1$.

New variables are:

AIJ	/COM25/TORCOF/ used to amend λ near blanked points
BLANK	/COM25/input/ invokes blanking option
BLANKIT	/argument/BLTST/ sends flag to *LNPTS* indicating status of current point
IB,JB,KB	/arguments/BLTST/ location of current point from *LNPTS* to *BLTST*
IBL	/COM25/input/ array containing blank flags on input grid file
ISTF,IENDF	/COM25/GETDATA/ saved input values of IST and IEND
MG1F,MG2F	/COM25/GETDATA/ saved input values of MG1 and MG2
NIPTSJ1	/COM25/LNPTS/ saved value of NIPTS on *j-1* line

5.4 Example Case: Access-to-Space

The blanking version of the SAGE code was developed to assist the Access-to-Space project, particularly the plume and base-flow computations. This example case consists of a multiple grid with a total of five overlaid grids enveloping the base and plume flow regions. Grid number three overlays the entire domain and contains many of the features that the new blanking version of the SAGE code must accommodate. It was therefore chosen as the test case of the adaption process. Figure 37 shows the constant *j=2* plane and two constant *i* planes, *i=48* and *i=20,* from grid #3 that is dimensioned (48×31×61). The arrows in the figure indicate the coordinate directions.

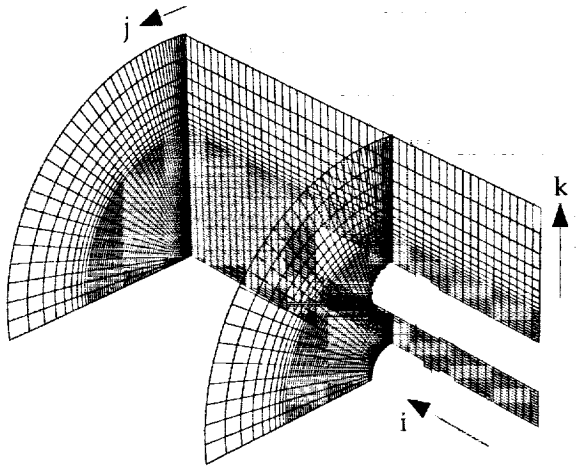

Figure 37. Grid 3 domain.

Adaption of a single j plane.

To examine the methodology of adapting around blanked regions, we first look at a single plane. Figures 38(a) and 38(b) show the initial grid and the interim Mach contours calculated on the constant *j=2* plane seen in Fig. 37.

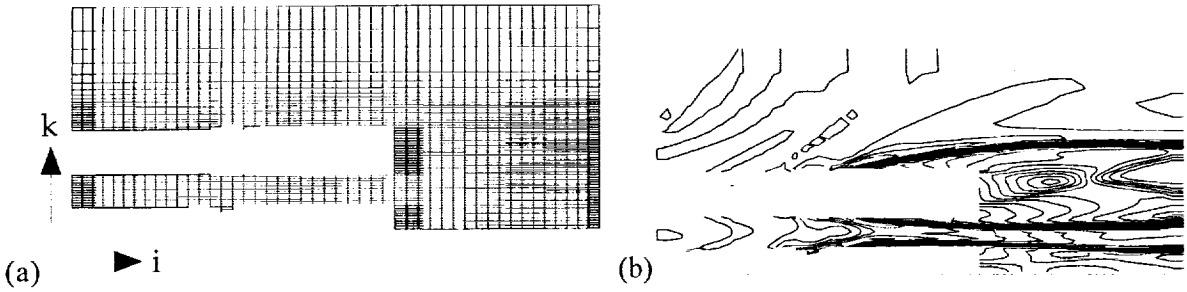

Figure 38. Plane j=2. (a) Initial grid; (b) initial Mach contours.

Since this was an interim solution, many of the flow features are not clearly developed. In addition, only 'noise' can be seen at the onset of the flow (*i* planes 1 through 16). As a result, the first adaption procedure was initiated at *i=16*. This example is complex because it contains a blanked region at a boundary, an inner blanked region, merging from two blanking areas to a full grid, and even some 'hanging' points/lines (points that are only singly attached to the grid) that can be seen halfway along the blanked boundaries.

The plane was adapted using the following user-parameter input file:

$namel mgrid=3,blank=.t.,ikplane=.t.,istep=.t., jst=2,jend=2,ist=16, indq=7, rdsmax=2.5,rdsmin=.4, clam(1)=.05, nedge=1,mgsteps=4 $

The *mgrid=3* parameter informs the SAGE code that the grid and solution files are in multiple grid format and that these adaption parameters apply to the third grid. The parameter *blank=.true.* ensures that the code reads the blank (IBL) array and tests for any blanked regions during the adaption of each line. The *ikplane* and *istep* parameters define the line-stepping and plane-stepping directions. The adaption domain is the *j=2* plane, *i* points range from 16 to 48, and all the *k* points are included (if the domain range is not input, physical boundaries are used as default). *Indq=7* indicates that Mach number is the adaption variable; *rdsmax* and *rdsmin* define the relative maximum and minimum mesh spacing; *clam(1)=.05* increases the spring force, ω, to produce a smoother adaption; *nedge=1* invokes the edge spacing control and *mgsteps=4* invokes the merging algorithm at *i=16* to moderate the discontinuity between the adapted and non-adapted region. All of these parameters are described in detail earlier in this user guide.

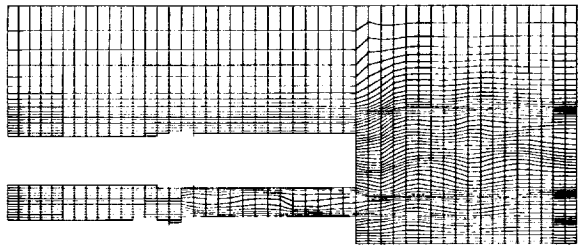

Figure 39. Adaption from left to right, starting at i=16.

This was a single-sweep adaption and the resulting grid is shown in Fig. 39. It can be seen that the upper left side of the grid has not been adapted. This is because the adaption algorithm can only handle one segment of a line, and since the line point count began at *k=1*, only the segment between the two holes was captured. In order to capture the upper segment, a second sweep is required, this time defining the adaption domain with *kst=61*, thus ensuring the line point-count algorithm starts looking for active points at *k=61*. The following two-sweep input parameter file produced the result shown in Fig. 40.

$namel mgrid=3, blank=.t., ikplane=.t.,istep=.t., indq=7,jst=2,jend=2, ist=16,kst=61, kend=1,rdsmax=2.5,rdsmin=.4, clam(1)=.05, nedge=1, mgsteps=4 $

$namel mgrid=3, blank=.t., ikplane=.t.,istep=.t., indq=7,jst=2,jend=2, ist=16, rdsmax=2.5,rdsmin=.4, clam(1)=.05, nedge=1, mgsteps=4 $

This adaption provides a quite satisfactory result: all points have been adapted to align with the flow features and no distortion of the blanked areas has occurred.

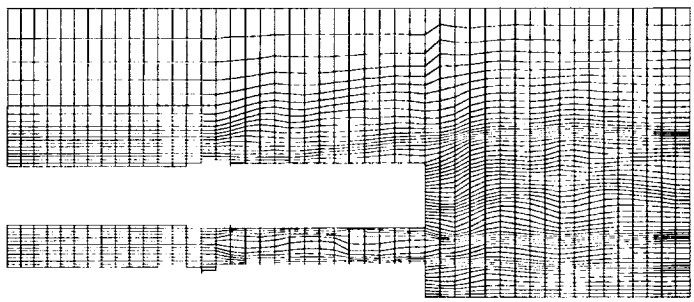

Figure 40. Two-sweep adaption.

Adaption of all planes.

Finally, Fig. 41 shows the complete adaption for all *j* planes. The input data set is almost unchanged: the single-plane restriction on *j* has been removed and a value for the torsion between planes has been input, giving

```
$name1  mgrid=3, blank=.t., ikplane=.t.,istep=.t., indq=7,ist=16, kst=61,kend=1,
        rdsmax=2.5,rdsmin=.4, clam(1)=.05, clam(2)=.05 nedge=1, mgsteps=4, $
 $name  mgrid=3, blank=.t., ikplane=.t.,istep=.t., indq=7,ist=16,rdsmax=2.5,rdsmin=.4
        clam(1)=.05, clam(2)=.05, nedge=1, mgsteps=4, $
```

The figure shows the *i=20* and *i=48* planes and the *j=2* and *jend* planes. All the flow features have been captured by the adaption.

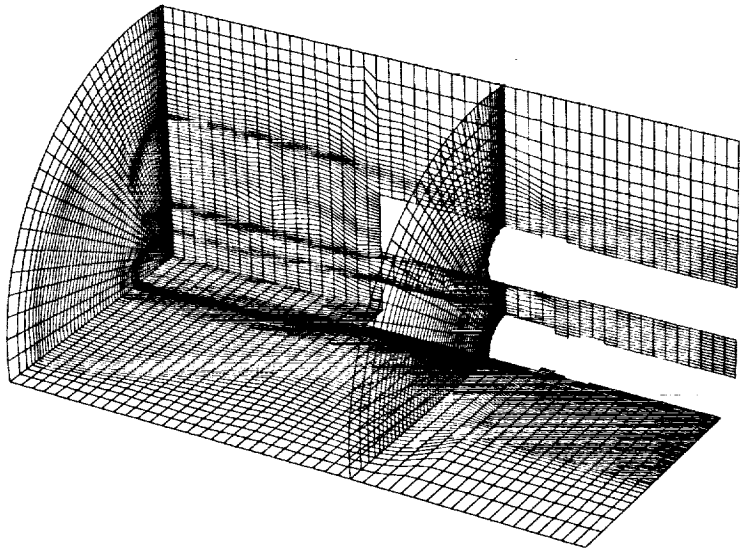

Figure 41. Full adaption of computational grid.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August1999	Technical Memorandum

4. TITLE AND SUBTITLE

The Self-Adaptive Grid codE, SAGE
Version 3

5. FUNDING NUMBERS

242-33-01

6. AUTHOR(S)

Carol B. Davies* and Ethiraj Venkatapathy†

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-99V0034

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA/TM–1999-208792

11. SUPPLEMENTARY NOTES

Point of Contact: Carol B. Davies, Ames Research Center, MS 230-2, Moffett Field, CA 94035-1000
(650) 604-6204
*Raytheon ITSS, Moffett Field, CA. †Thermoscience Institute, Eloret, Sunnyvale, CA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified — Unlimited
Subject Category 61 Distribution: Standard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The multi-dimensional self-adaptive grid code, SAGE, is an important tool in the field of computational fluid dynamics (CFD). It provides an efficient method to improve the accuracy of flow solutions while simultaneously reducing computer processing time. Briefly, SAGE enhances an initial computational grid by redistributing the mesh points into more appropriate locations. The movement of these points is driven by an equal-error-distribution algorithm that utilizes the relationship between high flow gradients and excessive solution errors. The method also provides a balance between clustering points in the high gradient regions and maintaining the smoothness and continuity of the adapted grid. The latest version, Version 3, includes the ability to change the boundaries of a given grid to more efficiently enclose flow structures and provides alternative redistribution algorithms.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Adaptive grids, Aerodynamics, CFD	73
	16. PRICE CODE: A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		